# Marco de Emulación Térmica para MPSoCs basado en NoC con DVFS

Proyecto Fin de Máster en Ingeniería de Computadores

Máster en Investigación en Informática, Facultad de Informática, Universidad Complutense de Madrid

**Autor: Emilio Martínez Pacheco**

**Director: David Atienza Alonso**

# Autorización de difusión

*El/la abajo firmante, matriculado/a en el Máster en Investigación en Informática dela Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Marco de emulación térmica para MPSoCs basados en NoC con DVFS ", realizado durante el curso académico 2008-2009 bajo la dirección de David Atienza Alonso en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.*

# Índice

# Resumen

El escalado en la tecnología impone un creciente stress sobre la temperatura en el diseño de circuitos digitales debido a la densidad de transistores, especialmente en sistemas de alta integración como son los Multi-Processor System-on-Chip (MPSoCs).

De esta manera, los diseños que tengan en cuenta la temperatura son necesarios y deben ser implementados en las fases tempranas del diseño de los MPSoCs para evitar iteraciones y retrasos en el desarrollo del producto final para el consumidor. En este proyecto presentamos una novel infraestructura de hardware que provee control térmico de las arquitecturas MPSoC, la cual está basada en explotar la interconexión de la NoC del sistema base como un componente activo que comunique y coordine entre los sensores de temperatuea colocados dentro del chip, de esta manera se puede monitorizar globalmente la temperatura actual del sistema. Entonces, una thermal management unit y unos clock frequency controllers son incluidos como parte de la infraestructura térmica activa basada en NoC para ajustar la frecuencia y el voltaje de los elementos de proceso de acuerdo con los requerimientos de temperatura de cada diseño de los MPSoC en runtime. Mostramos resultados experimentales de la aplicación de la propuesta infraestructura de manejo térmico basada en NoC activa para implentar unas efectivas políticas de control térmico de ámbito global para un MPSoC de 4 cores de la vida real, corriendo benchmarks de procesado de video de la vida real, emulados en un marco de emulación térmica basado en una FPGA. Además, debido al mejor balance térmico de nuestro propuesto control térmico basado en NoC activa, el rendimiento del MPSoC mejora al menos un 40% y consigue un 45% de ahorro de energía respecto a las aproximaciones de control térmico con DVFS local.

Palabras clave : MPSoC, Network-on-Chip, NoC, Sistemas empotrados, Diseños Thermal-aware, FPGA, Emulación

# Abstract

Technology scaling imposes an ever increasing temperature stress on digital circuit design due to transistor density, especially on highly integrated systems, such as Multi-Processor Systems-on-Chip (MPSoCs).
Therefore, temperatureaware design is mandatory and should be performed at the early design stages of MPSoCs to avoid iterations and delays in the deployment of final consumer products. In this proyect we present a novel hardware infrastructure to provide thermal control of MPSoC architectures, which is based on exploiting the NoC interconnects of the baseline system as an active component to communicate and coordinate between temperature sensors scattered around the chip, in order to globally monitor the actual temperature of the system. Then, a thermal management unit and clock frequency controllers are included as part of the active NoC-based thermal control infrastructure to adjust the frequency and voltage of the processing elements according to the temperature requirements of each MPSoC design at runtime. We show experimental results of the application of the proposed active NoC-based thermal management infrastructure to implement effective global temperature control policies for a real-life 4-core MPSoC, running real-life video processing benchmarks, emulated on an FPGA-based thermal emulation framework. Furthermore, due to the better thermal balancing of our proposed active NoC-based thermal control, the MPSoC performance improves almost 40% and achieves 45% energy savings with respect to local DVFS thermal control approaches.

Key words: MPSoC, Network-on-Chip, NoC, Embedded Systems, Thermal-Aware Design, FPGA, Emulation

# Motivación y puesta en contexto

## Problema de interconexión en los MPSocS

El problema del interconexionado se esta haciendo cada vez más crítico. En los sistemas antiguos bastaba con tener una jerarquía de buses, bastando muchas veces tener dos buses, uno para alta disponibilidad que conectara las memorias y dispositivos de alto rendimiento como tarjetas de video, y otro para dispositivos de bajas prestaciones, como pueden ser periféricos de interfaz humana. Sin embargo, al llegar el nuevo paradigma de integrar todos estos dispositivos en un único chip (MPSoC) la idea de la jerarquía de buses presenta varios problemas que no lo hacen adecuada para este tipo de sistemas. Estos son básicamente dos: ese tipo de interconexionado no escala adecuadamente con la tecnología de integración. Cuando operamos en tecnologías de 65 o 45 nm con las que se piensan integrar este tipo de sistemas, los largos cables que formarían los buses no pueden transmitir la información en un único ciclo de reloj, además forman grandes áreas dedicadas a la comunicación que empobrecen el ratio de integración. Por otra parte tenemos el problema del ancho de banda suministrado. En un sistema basado en buses el canal es compartido por todos los componentes, teniendo que disponer de algún módulo de arbitraje que coordine la comunicación entre los distintos componentes, en definitiva el uso de buses por parte de multitud de módulos como presentan los MPSoCs se traduce en unas pobres tasas de transferencias que se oponen radicalmente al fin con la que estos sistemas están construidos: dar soporte a aplicaciones multimedia de alto rendimiento. Por tanto podemos afirmar que las arquitecturas basadas en buses no escalan bien ni arquitectónica ni físicamente.

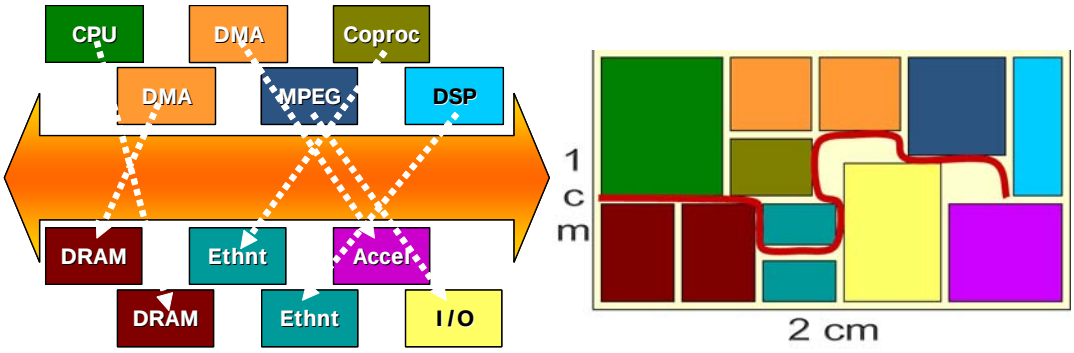



Figura 1. Antiguo paradigma basado en buses

Por tanto se necesita un cambio radical en la visión del interconexionado realizado on Chip, más teniendo en cuenta que se prevé que los chip estén cada vez más dominados por la interconexión. En este escenario hace aparición una nueva visión, un nuevo paradigma para interconectar los distintos módulos dentro del chip: las redes en chip o network on chip (NoCs). Una NoC es un conjunto de Switches y NI (Network interface) interconectados para dar servicio de comunicación a los distintos módulos.

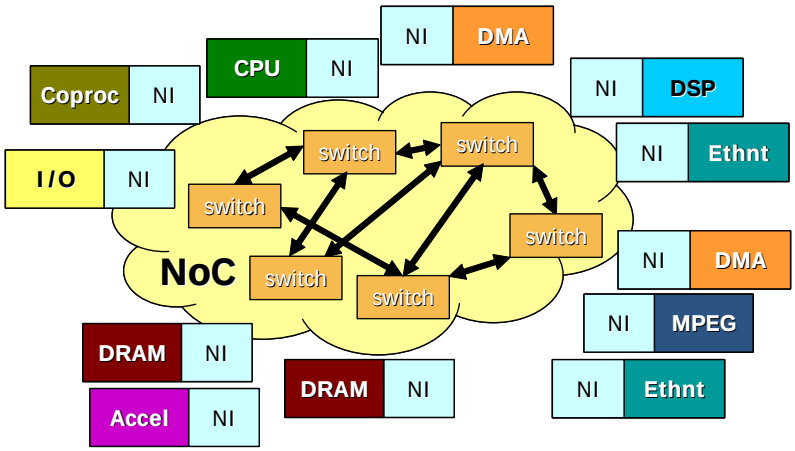



Figura 2. Nuevo paradigma basado en Noc

Las ideas están basadas en las redes de computadores, eso sí, teniendo en cuenta las limitaciones de implementación, ya que estamos hablando de

integrar esta red más todos los componentes o IP-cores en un mismo chip. Las características de las NoC pueden variar ampliamente.

La principal ventaja de las NoCs es su facilidad para escalar en dos sentidos. Primero, podemos dimensionar el diseño arquitectónico sin tener que hacer grandes cambios en el subsistema de interconexión, ya que las Noc presentan un patrón que puede ampliarse con facilidad. Segundo, podemos escalar (encoger) la tecnología de implementación sin ningún problema, ya que los retardos están segmentados entre los distintos puntos de interconexión entre switches, y no en grandes líneas como es el caso de los sistemas basados en buses.

## El Problema térmico

Es un hecho constatado que la temperatura es el gran reto en los modernos sistemas embebidos. Tenemos chips cada vez con más y más módulos. Si antes un chip integraba un procesador, hoy en día puede estar integrando un sistema multiprocesador con la memoria incluida, y distintos módulos aceleradores y de Entrada/Salida. La tecnología es cada vez más y más pequeña: de 10 micras con el Intel 4004 en el año 1971, pasamos a diseños de 350 nanómetros con el Pentium Pro en el año 1995, y ya más recientemente fabricaciones en 65 nanómetros en el Pentium 4. Esta disminución en el factor lambda produce un aumento en la densidad de las puertas que se traduce en un aumento en la densidad de calor, provocando los llamados Hot Spot, puntos con excesiva concentración de calor. Por último, tenemos diseños cada vez más rápidos, volviendo al ejemplo anterior, tenemos que el Intel 4004 funcionaba a 740 KHz mientras que el Pentium Pro lo hacia a 133 MHz, actualmente el Pentium 4 funciona a varios GHz. Por todos estos factores, impulsados en parte por la tan conocida ley de Moore, la temperatura está llegando a ser una de las asignaturas más importantes a la hora de diseñar chip. Tanto es así, que se están buscando diseños más “inteligentes” en lugar de más rápidos para conseguir mayores prestaciones, ahí esta el ejemplo de la línea Core Duo de Intel.

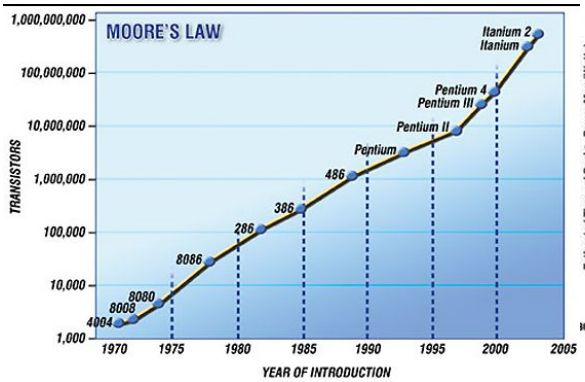



Figura 3. Ley de Moore

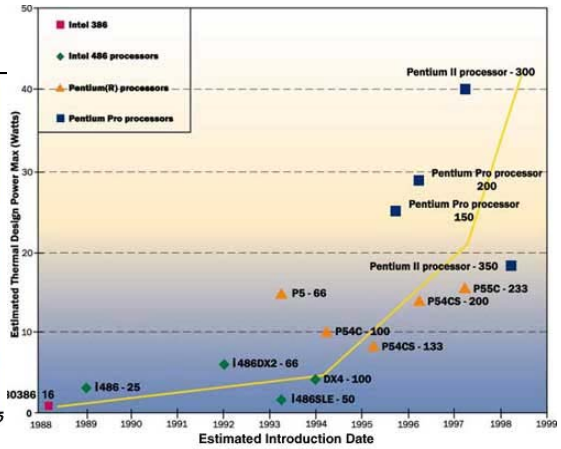



Figura 4. Disipación de potencia en procesadores Intel

## Una Posible Solución

En el presente trabajo vamos a sostener la idea de que una posible solución a ambos problemas citados anteriormente pasaría por la utilización de una NoC como sistema vertebrador a la hora de diseñar un MPSoC. De esta manera quedaría resuelto el problema de la interconexión entre los distintos módulos o IP-cores, y la ulterior inclusión en el diseño de más módulos. Esta es la función primordial con la que se creo el concepto de NoC, pero podemos aprovechar su naturaleza distribuida para llevar a cabo una recolección de información térmica, y recopilar dicha información en una unidad centralizada a la que

llamaremos Thermal Management Unit, TMU, que más tarde haga una planificación del comportamiento del sistema según una serie de parámetros que sean configurables en el sistema. En nuestro caso de estudio la TMU podrá controlar el punto de trabajo (Frecuencia/Voltaje) de cada uno de los cuatro procesadores que incluye el sistema. De esta manera podremos conseguir un comportamiento térmico controlado del sistema, y ajustarlo a distintas necesidades, como pueden ser típicamente: máximo rendimiento sin cruzar un cierto umbral térmico o bien, cumplir con una carga de trabajo con la mínima disipación de calor.

# Estado del Arte

El problema del control de la temperatura ha sido un asunto importante desde las dos últimas décadas, especialmente para sistemas de altas prestaciones. Dado el aumento de la importancia de análisis térmicos en etapas tempranas, varios modelos térmicos para systems-on-chip han sido desarrollados. A nivel físico, varios métodos pueden ser usados para modelar la transferencia de calor en el substrato. Finite-difference time domain [1], finite element [2] and Green-function [3] han sido la base de algoritmos aplicados en análisis térmico de sistemas on-chip. A nivel arquitectónico, [4] presenta un modelo térmico/potencia para arquitecturas superescalares. Todos estos modelos pueden ser usados en tiempo de diseño para estudiar el mejor emplazamiento de los componentes y el tamaño de los elementos enfriadores. En nuestro caso, nosotros usamos el modelo presentado para estimar con precisión la temperatura en run-time usando emulación del MPSoC basada en una FPGA, de esta manera podemos conseguir una emulación más rápida de largos intervalos de tiempo, determinantes en el estudio del comportamiento térmico de los MPSoC. Basada en las herramientas de modelado térmico citadas anteriormente, técnicas de Dynamic Thermal Management (DTM) han sido sugeridas para procesadores usando ambas adaptaciones arquitectónicas Dynamic Voltage Scaling (DVS), Dynamic Frequency and Voltage Scaling (DVFS), migración de tareas/actividades y técnicas de profiling. En [4] se propone usar teoría de control con feedback como manera de implementar técnicas adaptativas en la arquitectura del procesador. También, en [5] se realizan extensivos estudios empíricos de técnicas DTM (i.e. DVFS, fetch-toggling, throttling, y especulación) cuando el consumo de potencia del procesador cruza un determinado umbral (i.e. 24W). Sus resultados muestran

que escalado de frecuencia y DFS pueden ser muy ineficientes si su tiempo de invocación no es el apropiado. En [6] se presenta un algoritmo predictivo basado en frames y DTM, probado con una aplicación multimedia. Este algoritmo usa profiling para predecir el mayor rendimiento teórico manteniendo una configuración térmica dentro del límite de funcionamiento y cumpliendo los requerimientos de cada frame. También se han reconocido el impacto negativo sobre la fiabilidad del sistema de los gradientes de temperatura extremos [7]. Otras soluciones han sido propuestas con la visión de reducir la diferencia de temperatura en el chip mediante migración de procesos [8], esto está dirigido a reducir los picos de temperatura moviendo procesos intensivos entre distintas unidades replicadas. El mismo objetivo de homogenización de la temperatura es llevada a cabo en [7] en la etapa de diseño del SoC. Recientemente se han propuestos diseños que usan la NoC del sistema para tener en cuenta los problemas térmicos. En [9] se propone un algoritmo para el óptimo emplazamiento de los componentes on-chip para mejorar el balance térmico mientras se minimiza el coste en comunicación. Este trabajo es complementario a nuestra aproximación, y puede ser aplicado en fases tempranas del diseño cuando el floorplan de le chip final puede ser modificado, mientras nuestra aproximación entra dentro de las técnicas de control térmico en run-time. Finalmente, en [10], se propone homogenización de la temperatura a través reconfiguración dinámica de los hotspots. Para que esta estrategia pueda ser implementada, es necesario que el chip disponga de una región de hardware reconfigurable, si embargo nuestra propuesta no necesitaría este recurso.

# Entorno de Emulación Térmica

A continuación pasaremos a repasar de forma pormenorizada los distintos componentes que constituyen el prototipo diseñado utilizado para obtener los resultados, pero antes veremos un pequeño repaso a nivel general para hacernos una idea global.

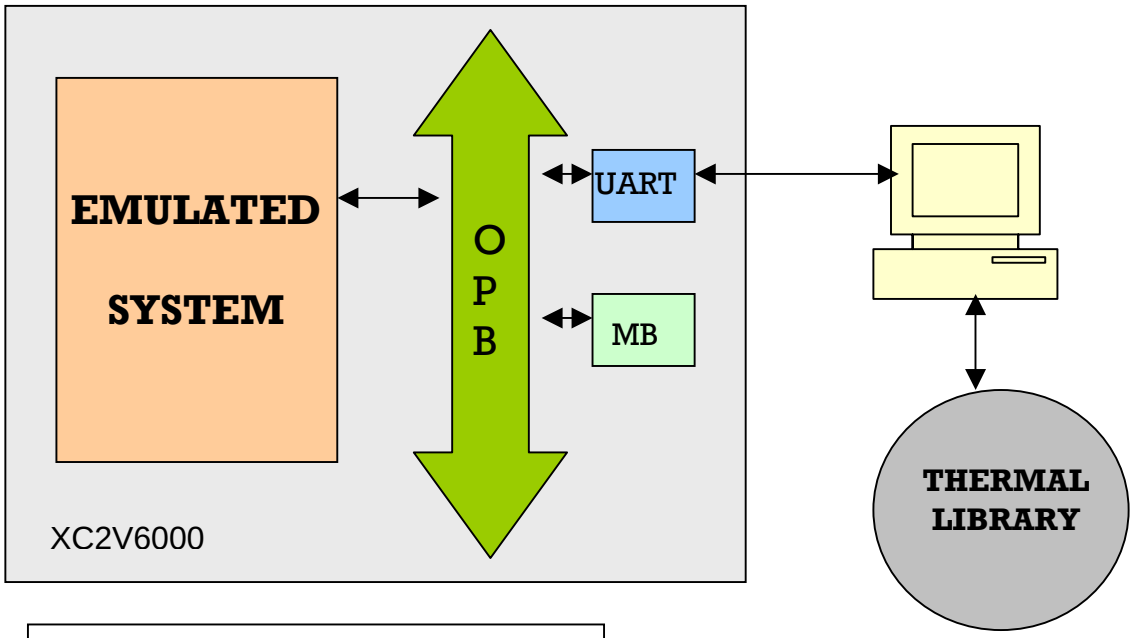



Figura 5. Visión global del sistema

En la figura 5 tenemos una visión de conjunto de todo el sistema de prototipado. Vemos que hay dos escenarios distintos, por una parte tenemos una FPGA XC2V6000 y por otra un computador de sobremesa. En estos dos escenarios se mueven tres actores distintos:

- El sistema emulado (MPSoC)
- El sistema supervisor (Extractor de estadísticas)
- La Librería térmica

## Sistema Emulado

Consiste en un sistema con 4 procesadores MicroBlaze que emulan el comportamiento del procesador ARM 9. Los procesadores están conectados mediante una NoC a una memoria privada cada uno y 2 memorias compartidas, una de ellas con la característica especial de ser accesible por el sistema supervisor, de manera que se pueda hacer la carga inicial de datos. También es accesible para los procesadores a través de la NoC la TMU, encargada de implementar la política de escalado de frecuencias. La emulación del DVFS es posible gracias a un módulo que ataca la entrada de reloj de los procesadores y que facilita la selección de un valor (1X, 1/2X, 1/4X ó 1/8X) de las frecuencias que multiplexa, de forma independiente para cada procesador.
Todo este esquema queda plasmado de forma esquemático en la siguiente figura.

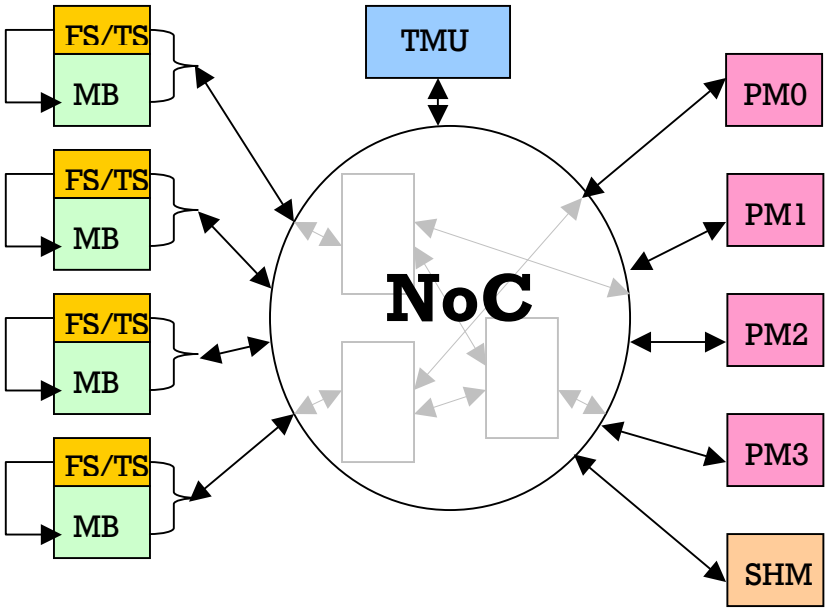



Figura 6. Esquema sistema emulado

# La NoC

La NoC usada en nuestro proyecto es una NoC de tipo heterogéneo diseñada para maximizar el acceso a la memoria privada de cada procesador en detrimento de los accesos a la memoria compartida. Mientras que la ruta del procesador hasta la memoria privada atraviesa un solo switch, para alcanzar la memoria compartida tendrá que atravesar 2, con el consecuente aumento en la latencia. Esta topología fue automáticamente diseñada con SunFloor un programa desarrollado en el Laboratoire des Systèmes Integres (LSI) de la Ecole Polytechnique Federale de Lausanne (EPFL).

La NoC contiene 3 switches de 5X5 (5 entradas y 5 salidas )completamente conectados, más 11 Network Interfaces (NI) 4 de ellos maestros y 7 esclavos.

La NoC implementa el sistema wormhole, por tanto los mensajes se irán propagando por los swtches abriendo camino con la cabeza del mensaje, y cerrando el canal de comunicación con la cola. Las rutas son determinísticas, son conocidas en el punto de envío y únicas. Las NoC generadas por SunFloor presentan la características de constituir un sistema GALS (Global Asynchronous Locally Synchronous) por tanto puede aceptar entradas en sus NI que se comunican por bus OCP con distintos dominios de reloj.

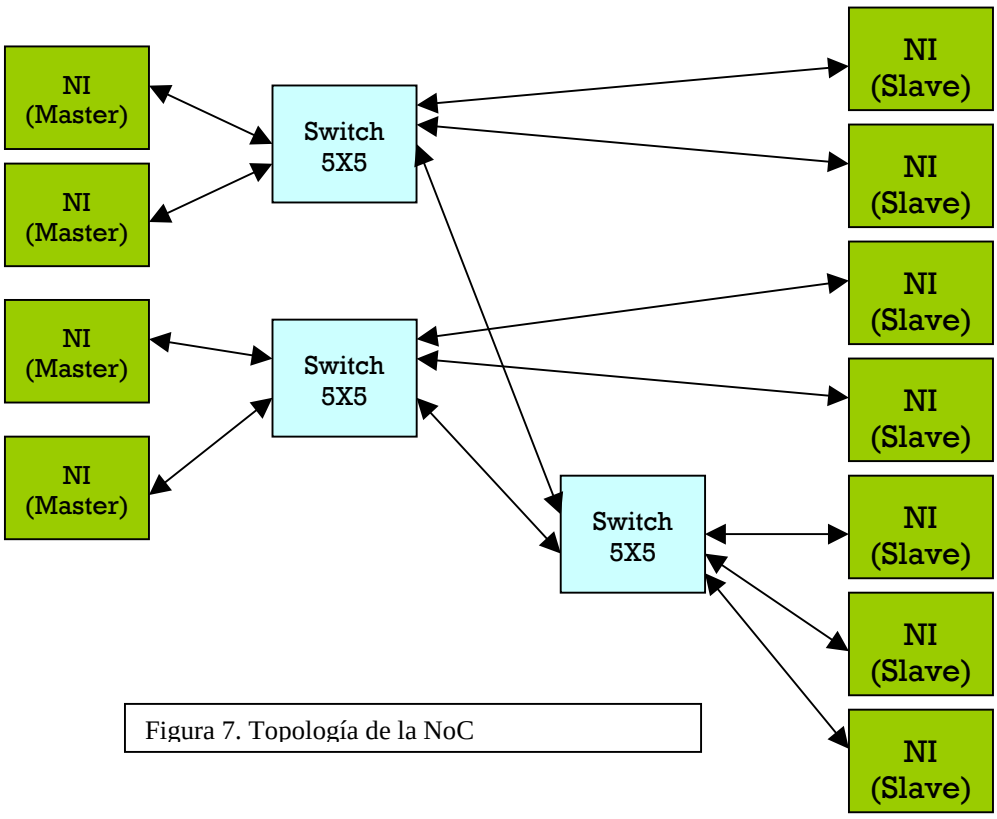



Figura 7. Topología de la NoC

## La Unidad de Control Térmico (TMU)

La TMU es el módulo encargado de centralizar toda la información térmica del sistema. En el sistema real estaría compuesto por un procesador ARM7 y su sistema de memoria, en la emulación se utiliza un MicroBlieze en su lugar. En el modelo térmico se tienen en cuenta las características del ARM7. La información térmica llega de los distintos nodos monitorizados (en nuestro caso los cuatro procesadores) a través de la NoC.

En el sistema real seguiría el siguiente procedimiento, con un sensor real para medir la temperatura:

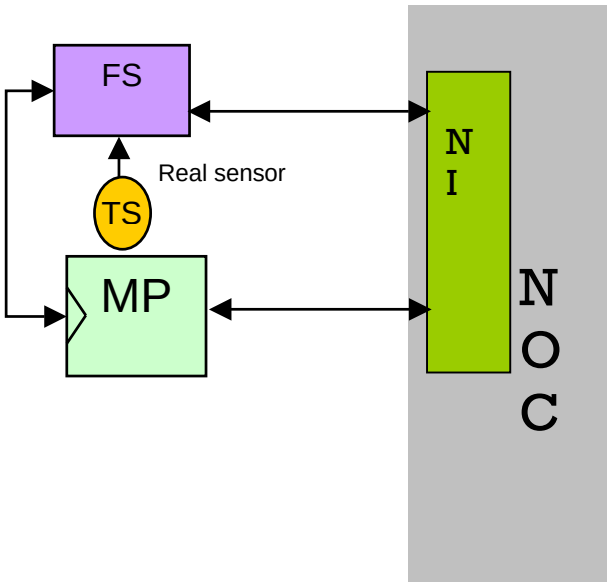



Figura 8. Thermal Sensor y Frequency Scaling (Sistema Real)

El módulo Frequency Scaling (FS) se encarga de seleccionar la velocidad de trabajo del Microprocesador (MP). Está conectado a la NoC mediante el mismo NI que el MP para poder recibir las órdenes de la TMU, que es quien realmente se encarga de fijar la frecuencia del procesador.

Sin embargo en el modelo emulado, el TS y el FS se fusionan en una sola unidad: TS/FS. El TS consiste en este caso en un registro que se carga con la información de la emulación térmica.

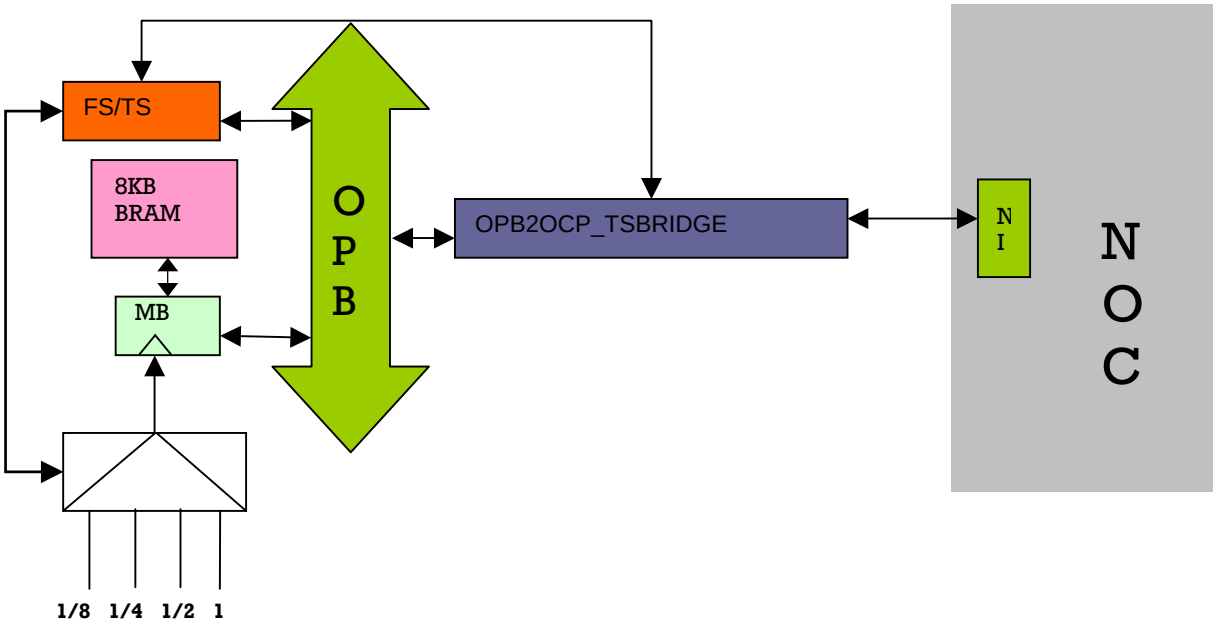



Figura 9. Thermal Sensor y Frequency Scaling (Sistema Emulado)

Hemos implementado soporte para varias políticas de manejo térmico en la TMU para validar la infraestructura térmica basada en NoC propuesta. En general, son necesarias políticas de frecuencias dinámicas en sistemas donde las características de la aplicación y cargas de trabajo pueden cambiar dinámicamente. Sin embargo, muchos MPSoCs están diseñados para ejecutar una serie limitada de aplicaciones, dependiendo del dominio de aplicaciones objetivo. Como ejemplo, en [11], los autores muestran que las NoCs para diseños de MPSoCs industriales típicamente soportan varios usos de caso o aplicaciones. En este tipo de MPSoCs, la carga de trabajo del sistema está bien caracterizado para cada una de las aplicaciones. Por lo tanto, usando los

tiempos específicos de procesamiento y idle de benchmarks multimedia, hemos computado la frecuencia máxima necesaria por la aplicación con la que la carga de trabajo es satisfecha a tiempo. Sin embargo estos valores precomputados no implican que el sistema sea óptimo térmicamente. De hecho, la frecuencia de operación de los procesadores debería ser dinámicamente variada durante la ejecución de diferentes aplicaciones, para un mejor ajuste a las restricciones térmicas del sistema y para obtener diseños con consumos de energía eficiente. En nuestro caso, incluimos la ejecución precaracterizada de aplicaciones en nuestra TMU, de esta manera, damos soporte para DVFS a nivel de aplicación obteniendo un diseño eficiente que satisface las estriccione térmicas.

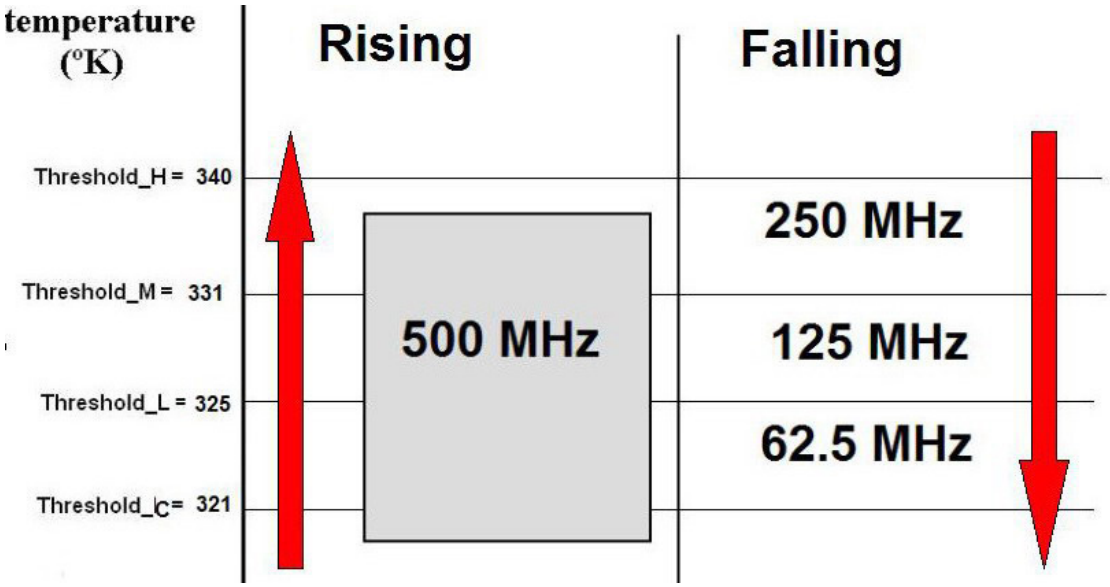



Figura 10. Umbrales de las políticas

Para demostrar la correcta funcionalidad del sistema de control térmico previamente descrito, hemos implementado tres diferente políticas en la TMU, de acuerdo con diferentes tipos de monitorización activa basada en la NoC del MPSoC. En todos estos casos el sistema multiprocesador puede operar a cuatro frecuencias/voltaje diferentes para cada procesador, estas son 500MHZ, 250MHz, 125Mhz y 62.5MHz, de acuerdo con las figuras encontradas

en “ARM 9 processing cores” [12] usada en nuestro modelo de MPSoC de 4 procesadores. Usando solamente estos cuatro puntos de operación desde el punto de vista del manejo hardware, se pueden hacer diferentes políticas térmicas basadas en DVFS que hagan el sistema mas estable desde el punto de vista de la temperatura, y que presente un comportamiento con menos oscilaciones térmicas abruptas (i.e. menos gradiente térmico y hot-spots), mediante la caracterización parcial off-line de las restricciones específicas de rendimiento de un conjunto de benchmarks multimedia que se ejecutan en un cierto intervalo de tiempo, similar a la aproximación sugerida en [6], y usando la NoC activa para predecir cuando una cierta aplicación a cambiado su fase de ejecución observando las transacciones de los diferentes Process Elements (PEs). Por lo tanto, debido a esta monitorización activa por parte de la NoC y el soporte existente de DVFS en los actuales MPSoCs [12],[13], los cambios de frecuencia y voltaje pueden realizarse en pocos ciclos de reloj, consiguiendo una adaptación de infraestructura muy rápida, con la cual se puede implementar complejas técnicas de control térmica con granularidad de “grano fino”. Hemos evaluado la efectividad de las capacidades de adaptación en run-time de la infraestructura propuesta de control térmico basada en NoC desarrollando tres políticas de control térmico para un caso de estudio real de 4 cores. El comportamiento de estas políticas es incremental, es decir, la segunda incluye el comportamiento de la primera y la tercera incluye el comportamiento de la segunda. De ese modo, observamos como el sistema de control térmico va mejorando cuando añadimos más conocimiento local y global sobre la temperatura actual y monitorizando las transacciones de los PEs en las políticas implementadas dentro de la TMU de nuestro sistema de control térmico basado en NoC.

**3-Threshold DVFS based on local temperature (DVFS Local):** Esta es una instancia de la política más frecuentemente implementada hoy en día para control térmico basado en hardware, en MPSoCs, donde la información térmica para cada PE, monitorizada por su respectivo TS, es usada como input a un algoritmo de control térmico. Para mantener la temperatura del MPSoC por debajo de un umbral de seguridad, una vez la temperatura del sistema pasa un cierto umbral, esta política cambia gradualmente ente diferentes valores cada

vez menores de puntos de operación DVFS (para prevenir grandes cambios en el DVFS) para reducir la temperatura de los PEs. Una vez la temperatura ha bajado por debajo de un umbral seguro, los PEs empiezan a incrementar su frecuencia de operación. En nuestro caso de estudio de un MPSoC con cuatro cores, usamos tres umbrales para obtener cuatro regiones diferentes de control térmico para decrementar la temperatura después de alcanzar un umbral de temperatura, como se ve en la figura 10. Esta figura nos muestra que hasta que un PE no supera los 340 grados Kelvin, trabaja a máxima velocidad (500 MHz). Entonces, por encima de esta temperatura la política cambia a 250MHz hasta que la temperatura de el PE está por debajo de 331 grados, que es cuando cambia a 125MHz hasta que la temperatura alcanza los 325 grados Kelvin que la frecuencia es puesta a 62.5MHz. Esta frecuencia se mantiene hasta que se alcanza el umbral de 321 grados Kelvin y el sistema vuelve a funcionar a plena potencia (500 MHZ) hasta que se vuelva a superar el umbral crítico.

**3-Threshold DVFS based on local temperature and local transactions monitoring (DVFS and local communication):** aparte de la temperatura local del procesador, como en la política previa, esta política de control térmico decide el punto DVFS de operación teniendo también en cuenta el número de transacciones de salida con la memoria local del procesador. De hecho, esta información adicional es usada para distinguir cuando la aplicación cambia entre las fases de espera y ejecución de los PEs en la implementación paralela del sistema empotrado multimedia que se ejecuta, y la frecuencia de los procesadores es adaptada de acuerdo con ello. Concretamente, si un procesador no efectúa suficientes transacciones en un cierto intervalo de tiempo, la frecuencia es decrementada, hasta un punto en que si no ocurren transacciones en un periodo (i.e. 10 mseg en nuestros experimentos), asumimos que el procesador no hará ninguna transacción en un significante periodo de tiempo, y su frecuencia es reducida al punto de operación mínimo (62.5MHz). Por otra parte, tan pronto como la NoC activa observa que comienza alguna transacción por parte del PE, la TMU comienza incrementando la frecuencia como en la política previa. Esta adaptación de la

frecuencia en las distintas fases de la aplicación es muy rápida, y sólo requieren pocos ciclos de reloj, por tanto no se observa penalización sustancial en el rendimiento del MPSoC final. La identificación de las fases en run-time de la aplicación para “afinar” las políticas de la TMU y el proceso de monitorización de la NoC, dependen de la aplicación/es que van a ser corridas en el MPSoC. En nuestro caso, este proceso fue hecho usando off-line profiling con fines ilustrativos solamente, y no es el tema de este trabajo. Existen aproximaciones de compiladores que hacen este trabajo de forma semi-automática off-line y podrían ser usados [6].

**3-Threshold DVFS with global temperature analysis and workload predictor (DVFS and global workload predictor):** Esta política de control térmico mejora la previa incluyendo información global adicional de la aplicación multimedia bajo estudio, la cual es monitorizada por la NoC, como un intento de evaluar cual podría ser el máximo beneficio para el control térmico que podía ser alcanzada usando monitorización del comportamiento del sistema a nivel global, conseguida por la infraestructura de NoC activa propuesta. Primero, usando un nivel de comprensión de “grano fino” de la simetría en la paralelización de la aplicación multimedia bajo estudio, se observo que uno de los procesadores era el encargado de colectar la información parcial procedente de la fase de procesado del resto de los cores. Por lo tanto, nuestra política de la TMU garantiza que la frecuencia de PE-2 a PE-4 nunca pueda ser superior que la frecuencia de PE-1 en ningún momento de la ejecución. De hecho debido a la regularidad parcial del procesado de video de la aplicación multimedia considerada, PE-1 determina la máxima frecuencia de todo el sistema, y desde el punto de vista balanceo térmico, es mejor correr los cores a la frecuencia menor posible para terminar su proceso cuando es necesitado para ser reagrupado para formar el output final en el PE-1, en lugar de terminar el cómputo de un cierto PE con un punto de operación DVFS mayor y después permanecer en estado idle. Segundo, de acuerdo con los ratios de transacciones de los PEs observados por la NoC, esta política adapta el punto de operación DVFS para cada PE de acuerdo con un tiempo predecido, estimado cuando PE-1 comenzará a recolectar la información de cada uno de los otros cores en cada iteración de la aplicación bajo estudio. Con

este fin, de acuerdo con los predichos deadlines requeridos por cada PE con respecto PE-1, la TMU elige al principio de cada ejecución un punto de operación DVFS para cada core. En nuestros experimentos, si la temperatura del sistema está por debajo de un valor crítico de temperatura, la TMU fijará 100% de potencia de cálculo al PE-1, el 75% para el PE-2, el 62.5% para el procesador 3 y el 50% para el procesador 4.

## Sistema supervisor

El sistema supervisor está compuesto por un procesador MicroBlaze, un módulo de entrada/salida tipo UART para comunicar la FPGA con el PC vía puerto COM, un temporizador y cuatro sniffers, todo esto interconectado mediante un bus OPB.

El funcionamiento del sistema es el siguiente: el sistema emulado corre por un tiempo determinado (10 ms en nuestros experimentos), los sniffers contabilizan todas las transferencias que se producen y su destino. Cuando el timer acaba la cuenta ,mediante la técnica de clock gating, congela el sistema emulado y comienza la tarea del sistema supervisor. El sistema supervisor descarga la información de los sniffers al PC mediante la conexión serie, es aquí en el PC donde se lleva acabo la simulación térmica del sistema emulado mediante la librería de simulación térmica. Una vez calculadas las temperaturas de los PE, estas son guardadas en los registros correspondientes que actual como sensores térmicos virtuales. Una vez las temperaturas están cargadas en los registros apropiados el timer vuelve a hacer la cuenta activando el funcionamiento del sistema emulado de nuevo.

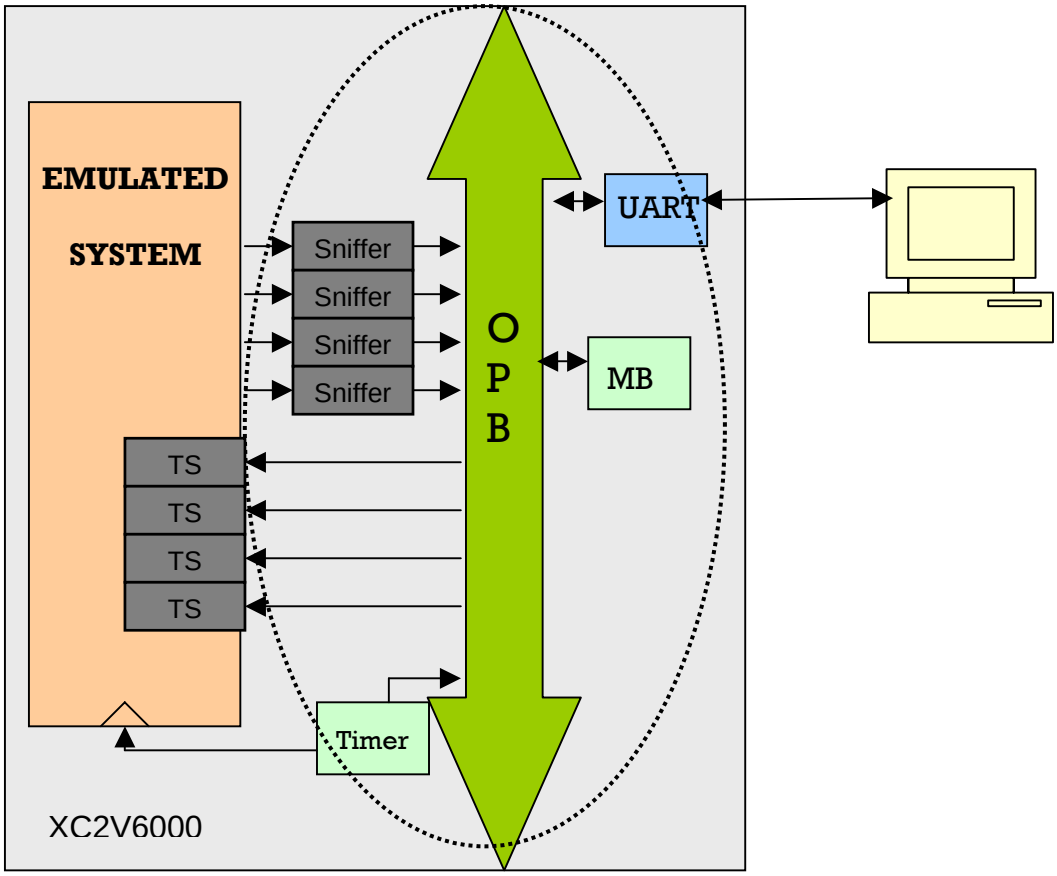



Figura 11. Detalle del sistema supervisor

## Librería térmica

Para calcular los valores de temperatura del floorplan usamos una librería térmica software diseñada en la universidad de Bolonia. El funcionamiento de dicha librería está basado en la analogía entre los sistemas RC y el comportamiento de la difusión del calor, más concretamente en una discretización de el caso general, basándose en un mallado para calcular las temperaturas. El floorplan se divide en celdas, y en cada una de estas celdas se aplica el modelo RC

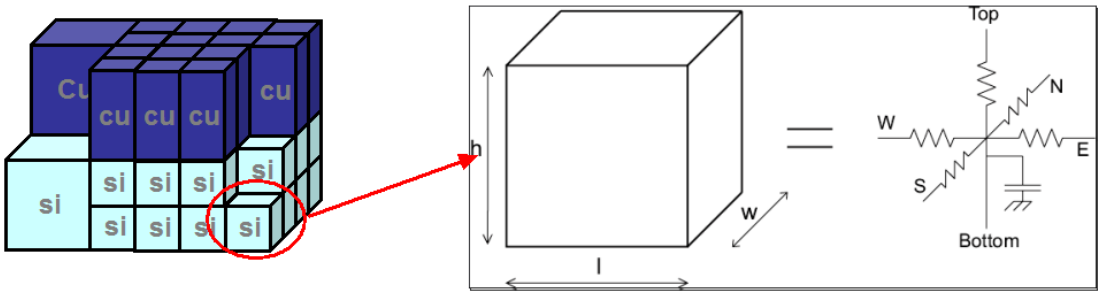



Figura 12. Mallado de la librería térmica

La librería distingue entre las diferentes áreas del chip como podemos ver en la siguiente figura.

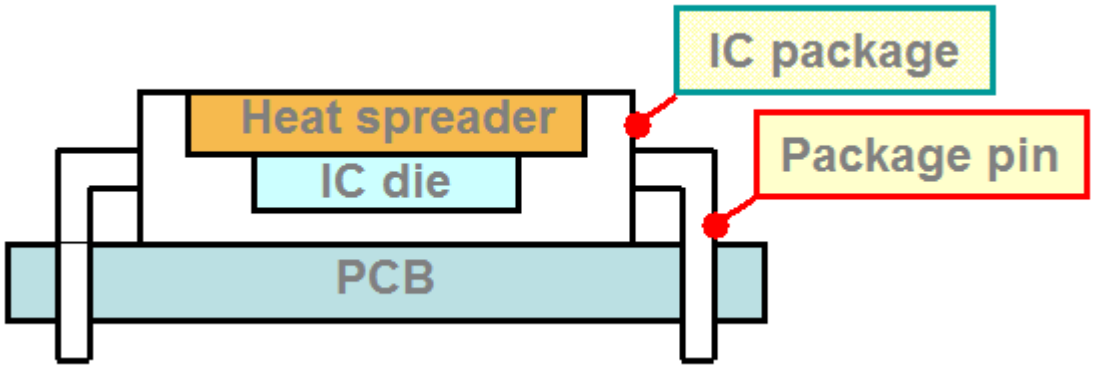



Figura 13. Distintas áreas del chip

El mallado que hemos utilizado en el MPSoC de nuestros experimentos es el siguiente:

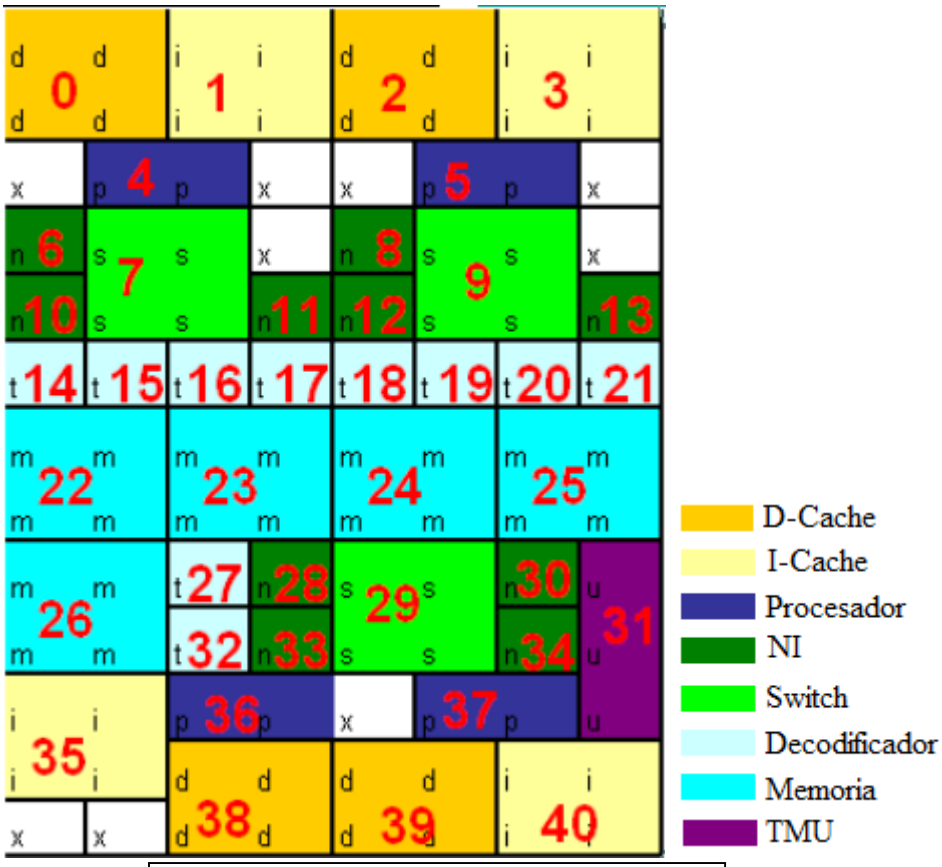



Figura 14. Floorplan del caso de estudio

# Instancia del Entorno de Emulación Térmica de MPSoCs

## Placas Usadas

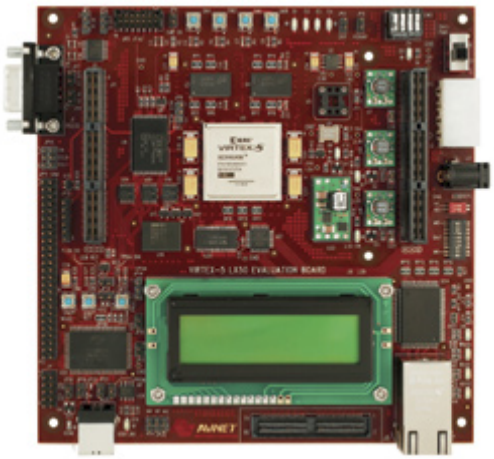
Para llevar a cabo nuestros experimentos hemos utilizado dos placas distintas. En una primera etapa realizada en el Laboratoire des Systèmes Integres (LSI) de la Ecole Polytechnique Federale de Lausanne (EPFL), se utilizó una placa ARM Emulation BaseBoard (HBI-0140D). En una segunda etapa ya en Madrid, en el DACYA de la UCM el proyecto se llevo a cabo en una placa de prototipado tipo Virtex V LX-50.

Vamos a ver en más detalle las características de cada placa.

- Principales características:
  - FPGA VirtexII V6000
  - Audio I/O
  - Ethernet
  - PCI
  - USB
  - 4 UARTs
- 2 slots para 2 ARM processor Tiles (4 procesadores por cada tile)

Figura 15. Placa de prototipado para ARM

En esta primera fase se hizo uso solamente de la FPGA de la placa, dejando para una fase posterior el uso de los ARM.



- Principales características:
  - Xilinx XC5VLX50-FF676 or XC5VLX110-FF676
  - Xilinx XCF32P Platform Flash
  - 10/100/1000 Ethernet PHY
  - 16 MB Flash
  - 64 MB DDR2 SDRAM
  - Cypress USB 2.0 controller
  - RS-232 serial port
  - 2 x 16 character LCD
  - 10-bit LVDS receive and transmit interfaces
  - Programmable LVDS clock generator
  - EXP expansion slot
  - BPI configuration support
  - 100 MHz oscillator (+ LVTTL oscillator socket)
  - System ACE interface
  - User LEDs, DIP switch, and push-buttons

Figura 16. Placa de prototipado Virtex 5 LX-50

## Instancia del MPSoC

El sistema propuesto en los primeros puntos ha sido emulado usando FPGAs Virtex de Xilinx. A continuación presentamos un esquema con la instanciación del sistema completo que lleva dentro la FPGA.

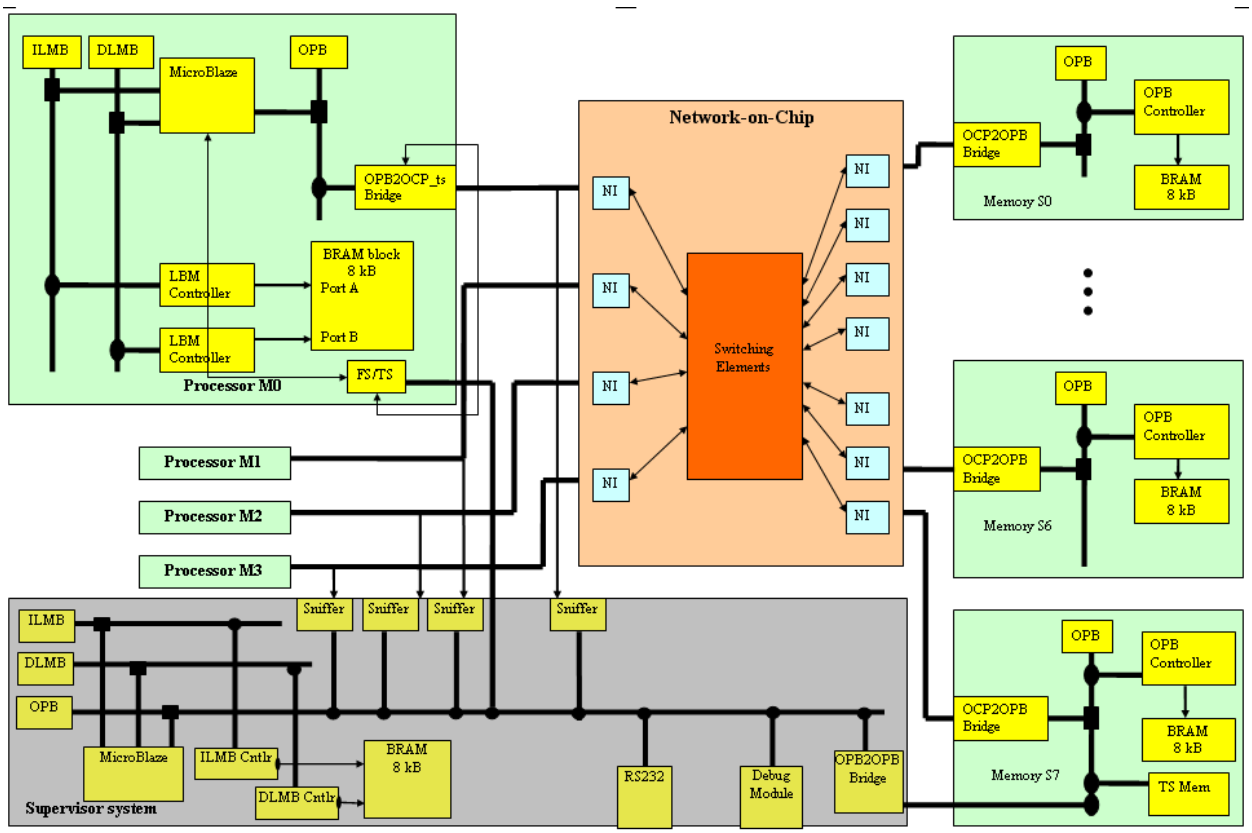



Figura 17. Instancia del MPSoC

Cada subsistema de procesador, que emula a un procesador ARM 9, está formado por un procesador MicroBlaze con un bus OPB al que va conectado el módulo OPB2OCP_ts Bridge, y los correspondientes buses ILMB y DLMB para

la conexión de una BRAM de 8KB. Además se conecta al procesador el módulo FS/TS (Frequency Scaling/Thermal) Sensor, encargado de seleccionar la frecuencia del procesador MicroBlaze y de actuar como registro para guardar la información térmica del procesador procesada por la librería térmica.

La NoC tiene la topología ya comentada, no representada aquí. Los NI tienen una interfaz OCP.

Cada subsistema de memoria contiene un módulo OCP2OPB bridge, un bus OPB, un módulo OPB Controller y una BRAM de 8KB. El subsistema de memoria número siete es un poco especial, y lleva además un módulo denominado TS Mem. Esto es así porque la TMU es emulada gracias a este subsistema y el Microbleze del sistema supervisor.

El subsistema supervisor está formado por un MicroBlaze, que también asumirá las funciones de la TMU, los buses ILMB y DLMB que conectan a una BRAM. Un bus OPB que contiene los siguientes módulos: 4 módulos de Sniffers, un módulo RS232, un módulo de Debug y un OPB2OPB bridge, que sirve para comunicar el MicroBlaze con el módulo TS Mem que es donde se podrán las frecuencias de funcionamiento de los subsistemas de procesadores, esta información tiene que cruzar la NoC para llegar al módulo FS/TS, así se emula más fielmente el funcionamiento del sistema emulado.

A continuación se muestra el mapeo en memoria de dos módulos construidos: sniffer y FS/TS.

## Mapeo del módulo Sniffer


Figura 18.Mapeo del módulo Sniffer

El módulo Sniffer consiste básicamente en 4 registros donde se guardan las transacciones producidas por el subsistema de procesador. Según se ve en la figura 18 el registro PM_WR contiene las escrituras en la memoria privada, el registro PM_RD las lecturas a la memoria privada, el registro SM_WR las escrituras a la memoria compartida y finalmente el registro SM_RD las lecturas a la memoria privada. Los Sniffers tienen una conexión al canal que comunica los módulos OPB2OCP_ts Bridge con los NI, que es un bus OCP.

## Mapeo del módulo FS/TS

El módulo FS/TS contiene 5 registros que explicaremos a continuación.

- El registro “Time Counter” indica en números de ciclos de reloj la periodicidad a la hora de enviar y recibir la información de los registros “Temperature” y “Frequency” a la memoria TS Mem. La información de “Temperature” se envía a TS Mem mientras que la información de “Frequency” se lee de ahí.
- El registro “Read Address” indica de que dirección de memoria tiene que leerse el registro “Frequency”.
- El registro “Write Address” indica en que dirección de memoria tiene que escribirse el registro “Temperature”.
- El registro “Temperature” contiene la temperatura del procesador calculada en la librería térmica.
- El registro “Frequency” contiene la frecuencia de funcionamiento del procesador.

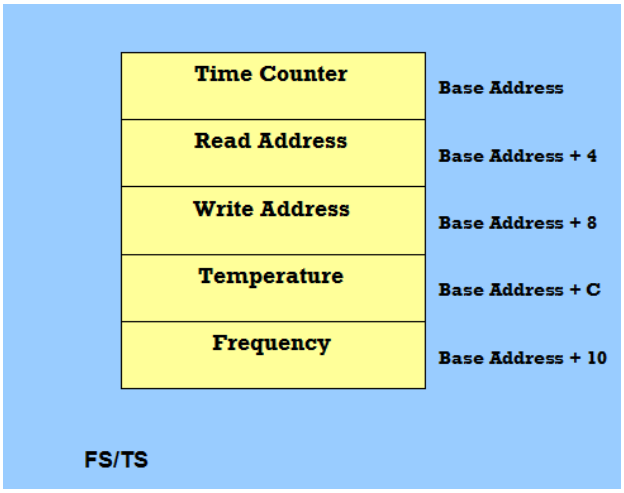



Figura 19.Mapeo del módulo FS/TS

## BenchMark usado

La aplicación multi-task para MPSoC usada para testear nuestra infraestructura de control térmica basada en NoC ha sido el software Visual Texture Coding (VTC) , software usado en el estándar MPEG-4 [14] para comprimir la información de la textura en modelos 3D foto realista. Como la textura en un modelo 3D es similar a una foto fija, la aplicación puede también ser usada para compresión de imágenes fijas. Está basado en la codificación “discrete wavelet transform, scalar quantization, zero-tree”. La realización del software requiere de alrededor de 10000 líneas de código C++. En particular, hemos usado una implementación paralela de este benchmark basado en multiplicaciones bidimensionales complejas 32X32 dividido en 4 procesadores, 8 filas por procesador, entre “matching windows” de dos frames consecutivos.

# Resultados experimentales

De nuestros experimentos, la primera conclusión es la evolución térmica en run-time del caso de estudio para el MPSoC citado mientras ejecuta múltiples instancias de la aplicación software paralela VTC.

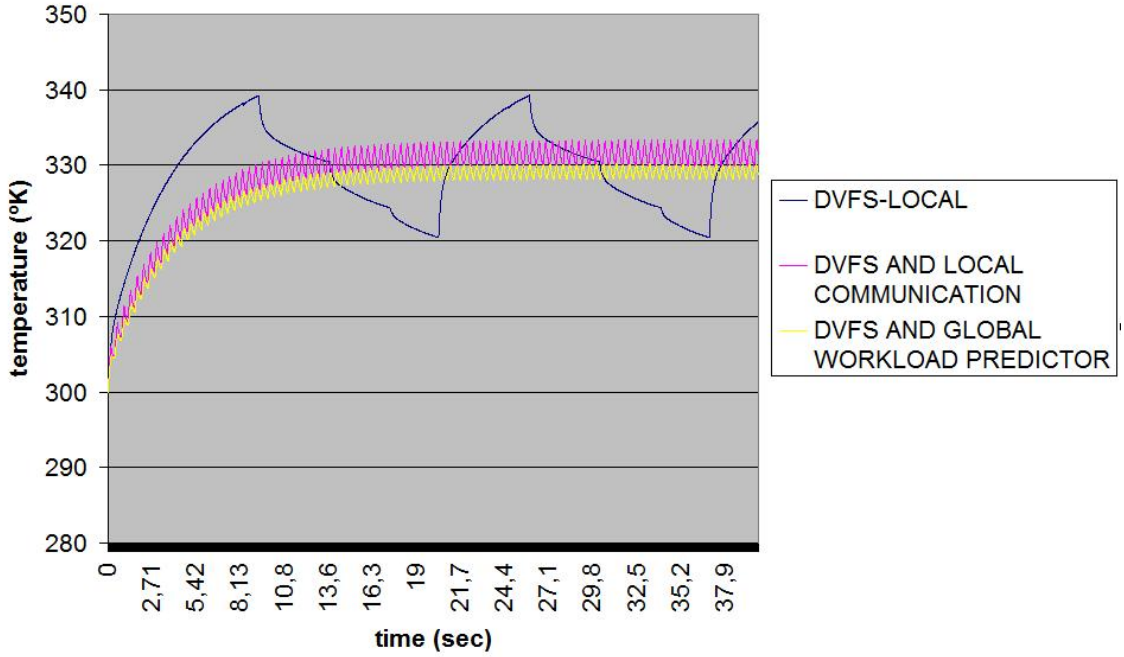



Figura 20. Media de temperatura del sistema para cada política

Como se muestra en la figura 20, los dos esquemas de control térmica basados en la propuesta NoC activa que monitoriza con la TMU global consiguen unas transiciones mucho más suaves en variaciones de temperatura en el “die” con respecto a las políticas de DVFS locales basadas en hardware del state of the art, esto se traduce en un efecto de balanceado térmico en el comportamiento del MPSoC, i.e., la temperatura se estabiliza en condiciones normales de trabajo a aproximadamente 328 grados Kelvin. Por lo tanto, la NoC activa consigue una mejora en la temperatura de trabajo en el MPSoC de 10 grados de media en comparación a la política de control térmico de DVFS local, lo cual posteriormente puede significar una mejora en la fiabilidad del sistema, ya que este es un factor que decrece exponencialmente con la temperatura de trabajo [15],[4].

Además, gracias al comportamiento térmico más balanceado mejorado por la NoC activa con TMU global, el MPSoC no hace frecuentes transiciones entre extremos puntos de funcionamiento de DVFS (500MHZ y 62.5MHz), como sí lo hace la política de control térmico de DVFS local.

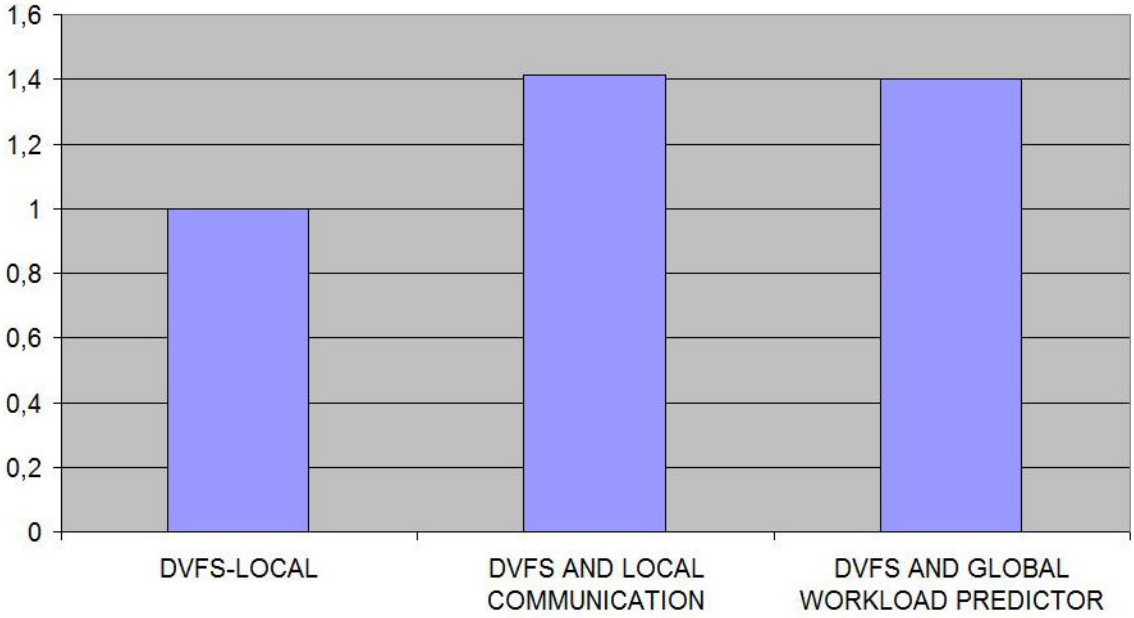



Figura 21. Rendimiento obtenido para las distintas políticas de control térmico (normalizado según DVFS local)

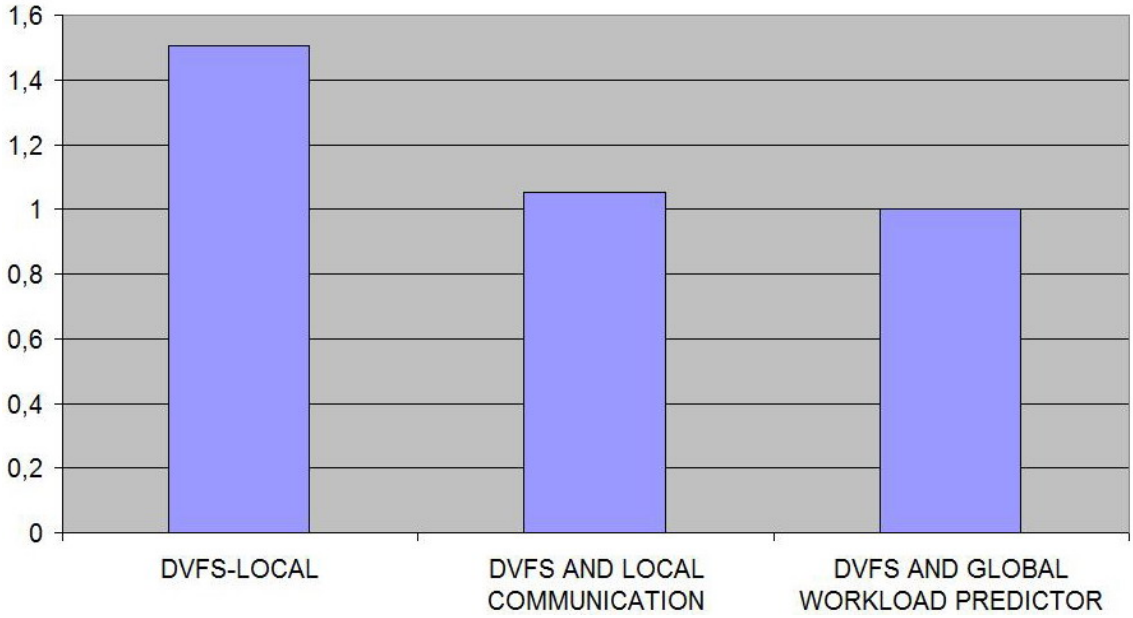



Figura 22. Consumo de energía obtenido para las distintas políticas de control térmico (normalizado según DVFS AND GLOBAL WORKLOAD PREDICTOR)

Así, según se muestra en las figuras 21 y 22, el MPSoC de 4 cores experimenta un nivel de rendimiento medio significativamente mayor (aproximadamente un 40%) y un ahorro considerable en el consumo de energía (casi 50%) usando la política de control térmica con información global y predictor de carga de trabajo, respecto a la política local.

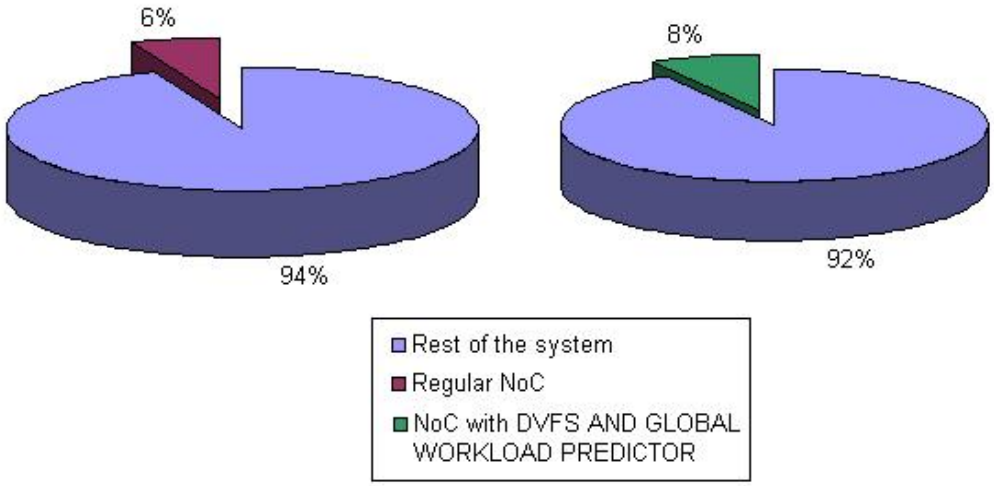



Figura 23. Comparación de consumo de energía entre la NoC básica y la NoC con control activo

Finalmente, presentamos en la figura 23 el porcentaje de energía consumida por la infraestructura de la NoC activa con respecto a la NoC básica usando tres switches 5X5 para el MPSoC de 4 cores. Como se muestra en la figura 23, la NoC activa con una TMU global solo genera un despreciable 2% de sobrecarga en el consumo de energía con respecto a la NoC básica. Esto es debido al hecho de que los mensajes de control térmico generados por la NoC activa requieren un ancho de banda de la NoC muy limitado, porque la dinámica térmica de los materiales es muy lenta (en el orden de milisegundos) con respecto a el volumen y frecuencia de los mensajes regulares de los componentes del MPSoC básico. Por lo tanto, la información térmica usada por la TMU para tomar sus decisiones de control térmico, así como los mensajes de control térmico mandados por la TMU a los PEs (Processor Element), pueden ser incluidos en el existente ancho de banda del diseño de la NoC

básica. Estos resultados ilustran la aplicabilidad de la propuesta infraestructura de NoC activa a los esquemas de control térmico de las arquitecturas de MPSoC de la vida real.

# Conclusiones

Debido a las mejoras en el proceso tecnológico, las arquitecturas de MPSoC se están convirtiendo en la solución para proveer el requerido rendimiento de las aplicaciones de usuario (e.g., procesado de video, juegos 3D) que operan en los sistemas de consumo. Sin embargo, el escalado del proceso de la tecnología impone importante stress en la temperatura en el diseño de circuitos debido a la creciente densidad de potencia. Por lo tanto, el desarrollo de eficientes mecanismos de control térmico es una pieza clave a la hora de diseñar la arquitectura de los MPSoCs. En este trabajo hemos presentado una infraestructura hardware de control térmico que saca partido al concepto de una NoC activa que monitorice la actividad de la comunicación y la evolución de la temperatura de los componentes básicos del MPSoC (procesadores, memorias, etc...) e incluye una TMU centralizada que provee de políticas de control térmico basadas en DVFS global. El propuesto flujo de diseño y la implementación de la NoC activa reutilizan los NI existentes para transmitir los mensajes de temperatura, como pequeños mensajes de control, en la infraestructura de interconexionado, lo cual resulta en una despreciable sobrecarga del ancho de banda para implementar el control térmico. Nuestros experimentos emulando un MPSoC industrial de 4 cores en nuestro marco de emulación térmica basado en una Xilinx Virtex-V FPGA, ha mostrado que la aproximación de la NoC activa consigue una mejora mayor a los 10 grados de reducción en la temperatura de trabajo media respecto a las técnicas de control térmicas basadas en DVFS local. Además, como consecuencia del mejor control térmico, el rendimiento del MPSoC mejora sobre el 40%, siguiendo respetando las restricciones de temperatura, y se consigue más del 45% de ahorro de energía.

# Publicaciones Realizadas

- Inducing Thermal-Awareness in Multi-Processor Systems-on-Chip Using Networks-on-Chip", Emilio Martinez, David Atienza, Proc. of IEEE Annual Symposium on VLSI 2009 (ISVLSI), Tampa, USA, IEEE Computer Society Press,ISBN: 978-0-7695-3684-2/09, DOI 10.1109/ISVLSI.2009.25, pp. 187 – 192, 13-15 May 2009.

# Referencias

[1] T.-Y. Wang and C.-P. Chen. 3-d thermal-adi: A linear-time chip level transient thermal simulator. IEEE Transactions on Computer-Aided Designs (T-CAD), 21(12), December 2002.

[2] B. Goplen and S. Sapatnekar. Efficient thermal placement of standard cells in 3d ics using a force directed approach. In Proceedings of the International Conference on Computer Aided Design (ICCAD), pages 86–89, San Jose, CA, Nov 2003.

[3] Y. Zhan and S. Sapatnekar. Fast computation of the temperature distribution in vlsi chips using the discrete cosine transform and table look-up. In Proceedings of the Asia/South Pacific Design Automation Conference (ASPDAC), pages 87–92. ACM Press, 2005.

[4] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. Temperature-aware microarchitecture: Modeling and implementation. Transaction on Architectures
and Code Optimizations (TACO), 1(1):94–125, 2004.

[5] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA), pages 171–182, Monterrey, Mexico, January 2001.

[6] J. Srinivasan and S. V. Adve. Predictive dynamic thermal management for multimedia applications. In Proceedings of the 17th annual international conference on Supercomputing (ICS03), pages 109–120, New York, NY, USA, June 2003. ACM Press.

[7] T. Sato, J. Ichimiya, N. Ono, K. Hachiya, and M. Hashimoto. On-chip thermal gradient analysis and temperature flattening for soc design. In ASP-DAC ’05: Proceedings of the 2005 conference on Asia South Pacific design automation, pages 1074–1077, New York, NY, USA, 2005. ACM Press.

[8] S. Heo, K. Barr, and K. Asanovi;. Reducing power density through activity migration. In In Proceedings International Symposium on Low Power Design (ISLPED), pages 217–222, New York, NY, USA, August 2003. ACM Press.

[9] W. Hung, C. Addo-Quaye, T. Theocharides, Y. Xie, N. Vijaykrishnan, and M. J. Irwin. Thermal-aware ip virtualization and placement for networks-on-chip architecture. In ICCD ’04: Proceedings of the IEEE International Conference on Computer Design (ICCD’04), pages 430–437, Washington, DC, USA, 2004. IEEE Computer Society.

[10] G. M. Link and N. Vijaykrishnan. Hotspot prevention through runtime reconfiguration in network-on-chip. In Proceedings of DATE, volume 01, pages 648–649, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[11] S. Murali and G. D. Micheli. Sunmap: a tool for automatic topology selection and generation for nocs. In DAC ’04: Proceedings of the 41st annual conference on Design automation, pages 914–919, New York, NY, USA, 2004. ACM Press.

[12] i.mx31 multimedia applications processors, 2003. www.freescale.com/imx31.

[13] Cradle technologies: Multi-core dsps for ip network surveillance, 2005. www.cradle.com/.

[14] I. S. et al. Scalable wavelet coding for synthetic and natural hybrid images. IEEE Transactions on Circuits and Systems For Video Technologhy, 9(2), March 1999.

[15] T. Simunic, K. Mihic, and G. D. Micheli. Optimization of reliability and power consumption in systems on a chip. In L. N. in Computer Science, editor, Proceedings of 15th Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS ’05), volume 3728, pages 237 – 246, August 2005.

# Apéndices

## Código sistema supervisor

```
// Located in: microblaze_0/include/xparameters.h
#include "xparameters.h"

#include "stdio.h"

#include "xutil.h"

//====================================================

#define CLK_BY_1  0x100
#define CLK_BY_2  0x040
#define CLK_BY_4  0x004
#define CLK_BY_8  0x001

#define TMU_POL   7

#define THRESHOLD_H     340000  //mK
#define THRESHOLD_L     325000  //mK
#define THRESHOLD_M     331000
#define THRESHOLD_C     321000

#define THRESHOLD_H0    343000  //mK
#define THRESHOLD_H1    338000  //mK
#define THRESHOLD_H2    333000  //mK

#define THRESHOLD_L0    320000  //mK
#define THRESHOLD_L1    325000  //mK
#define THRESHOLD_L2    330000  //mK

//Matrix multiplication constants
#define MSIZE 32
#define PROC_NUM 4

#define NO_COMMAND            0x00000000
#define MUL_MATRIX            0x0000FFFF
#define MOVE_TO_COM           0xFFFF0000
#define MOVE_FROM_COM   0xFFFFFFFF


#define NUM_TRANS             80 //4176 show the matrix resoult


//Sniffers Mapping
unsigned volatile int* sniffer0_PM_WR    = 0xf0000000; //Private
Memory (write accesses)
unsigned volatile int* sniffer0_PM_RD    = 0xf0000004; //Private
Memory (read accesses)
unsigned volatile int* sniffer0_SM_WR    = 0xf0000008; //Shared Memory
(write accesses)
unsigned volatile int* sniffer0_SM_RD    = 0xf000000C; //Shared Memory
(read accesses)
```

```
unsigned volatile int* sniffer1_PM_WR    = 0xf0010000; //Private
Memory (write accesses)
unsigned volatile int* sniffer1_PM_RD    = 0xf0010004; //Private
Memory (read accesses)
unsigned volatile int* sniffer1_SM_WR    = 0xf0010008; //Shared Memory
(write accesses)
unsigned volatile int* sniffer1_SM_RD    = 0xf001000C; //Shared Memory
(read accesses)

unsigned volatile int* sniffer2_PM_WR    = 0xf0020000; //Private
Memory (write accesses)
unsigned volatile int* sniffer2_PM_RD    = 0xf0020004; //Private
Memory (read accesses)
unsigned volatile int* sniffer2_SM_WR    = 0xf0020008; //Shared Memory
(write accesses)
unsigned volatile int* sniffer2_SM_RD    = 0xf002000C; //Shared Memory
(read accesses)

unsigned volatile int* sniffer3_PM_WR    = 0xf0030000; //Private
Memory (write accesses)
unsigned volatile int* sniffer3_PM_RD    = 0xf0030004; //Private
Memory (read accesses)
unsigned volatile int* sniffer3_SM_WR    = 0xf0030008; //Shared Memory
(write accesses)
unsigned volatile int* sniffer3_SM_RD    = 0xf003000C; //Shared Memory
(read accesses)
//-------------------------------------------------------------------
---------


// Thermal Sensors Mapping

//           ----------------------
//          |      Counter         |  Ts_BaseAdrress, determine the
WR/RD period  (cycles)
//     ----------------------
//          |     Read_Adrress        |  Ts_BaseAdrress + 4,the TS
reads from here the frequency
//           ----------------------
//          |     Write_Adrress       |  Ts_BaseAdrress + 8, the TS
writes here the temperature
//           ----------------------
//          |     Temperature         |  Ts_BaseAdrress + C
//           ----------------------
//          |     Frequency           |  Ts_BaseAdrress + 10
//           ----------------------
/*
            Freq Register           Frequenzy
            -------------     ---------
                  0x200                       clk_dcm_53
                  0x100                       clk_dcm_48
                  0x080                       clk_dcm_80/2
                  0x040                       clk_dcm_53/2
                  0x020                       clk_dcm_24
                  0x010                       clk_dcm_80/4
                  0x008                       clk_dcm_16
                  0x004                       clk_dcm_56/4
                  0x002                       clk_dcm_12
                  0x001                       clk_dcm_80/8
*/
```

```
unsigned volatile int* ts0_Counter =     0x30000000;
unsigned volatile int* ts0_RD_Addr =     0x30000004;
unsigned volatile int* ts0_WR_Addr =     0x30000008;
unsigned volatile int* ts0_Temp =        0x3000000C;
unsigned volatile int* ts0_Freq =        0x30000010;

unsigned volatile int* ts1_Counter =     0x30010000;
unsigned volatile int* ts1_RD_Addr =     0x30010004;
unsigned volatile int* ts1_WR_Addr =     0x30010008;
unsigned volatile int* ts1_Temp =        0x3001000C;
unsigned volatile int* ts1_Freq =        0x30010010;

unsigned volatile int* ts2_Counter =     0x30020000;
unsigned volatile int* ts2_RD_Addr =     0x30020004;
unsigned volatile int* ts2_WR_Addr =     0x30020008;
unsigned volatile int* ts2_Temp =        0x3002000C;
unsigned volatile int* ts2_Freq =        0x30020010;

unsigned volatile int* ts3_Counter =     0x30030000;
unsigned volatile int* ts3_RD_Addr =     0x30030004;
unsigned volatile int* ts3_WR_Addr =     0x30030008;
unsigned volatile int* ts3_Temp =        0x3003000C;
unsigned volatile int* ts3_Freq =        0x30030010;
/*
-------------------------------------------------------------
  End Thermal Sensors
-------------------------------------------------------------
*/

unsigned volatile int* UART_Receive = 0x40600000;
unsigned volatile int* UART_Transmit = 0x40600004;
unsigned volatile int* UART_SR = 0x40600008;

unsigned volatile int* ts_timer = 0x30040000;

unsigned volatile int* freq0 = 0x21FF0000;
unsigned volatile int* freq1 = 0x21FF0008;
unsigned volatile int* freq2 = 0x21FF0010;
unsigned volatile int* freq3 = 0x21FF0018;

unsigned volatile int* COM = 0x21000000;


unsigned int outputData[1044];

unsigned int temperatures[4];

unsigned int freqTable[4][4];

unsigned int cont[4];

int main (void) {

      int i=0,j=0;
      char* ptr_output = (char*)outputData;
      char* ptr_temp = (char*)temperatures;
      int contTemp;
      int contMod4;
      int numTrans[4];
      int freqAnt[4];
```

```
      int freqMitad;

      int cooling = 0;
   /*
    * Enable and initialize cache
    */
   #if XPAR_MICROBLAZE_0_USE_ICACHE
      microblaze_init_icache_range(0,
XPAR_MICROBLAZE_0_CACHE_BYTE_SIZE);
      microblaze_enable_icache();
   #endif

   #if XPAR_MICROBLAZE_0_USE_DCACHE
      microblaze_init_dcache_range(0,
XPAR_MICROBLAZE_0_DCACHE_BYTE_SIZE);
      microblaze_enable_dcache();
   #endif

   //print("-- Entering main() --\r\n");

      cont[0] = 0;
      cont[1] =0 ;
      cont[2] = 0;

      //Initialize Frequncies
      *freq0 = CLK_BY_1;
      *freq1 = CLK_BY_1;
      *freq2 = CLK_BY_1;
      *freq3 = CLK_BY_1;
      freqAnt[0] = CLK_BY_1;
      freqAnt[1] = CLK_BY_1;
      freqAnt[2] = CLK_BY_1;
      freqAnt[3] = CLK_BY_1;


      freqTable[0][0] = CLK_BY_1;
      freqTable[0][1] = CLK_BY_2;
      freqTable[0][2] = CLK_BY_4;
      freqTable[0][3] = CLK_BY_8;
      freqTable[1][0] = CLK_BY_2;
      freqTable[1][1] = CLK_BY_2;
      freqTable[1][2] = CLK_BY_4;
      freqTable[1][3] = CLK_BY_4;
      freqTable[2][0] = CLK_BY_4;
      freqTable[2][1] = CLK_BY_4;
      freqTable[2][2] = CLK_BY_8;
      freqTable[2][3] = CLK_BY_8;
      freqTable[3][0] = CLK_BY_1;
      freqTable[3][1] = CLK_BY_1;
      freqTable[3][2] = CLK_BY_2;
      freqTable[3][3] = CLK_BY_2;

      //Initialize Termal sensors
      *ts0_Counter =          0x10000;
      *ts0_RD_Addr =    0x21FF0000;
      *ts0_WR_Addr =    0x21FF0004;
      *ts0_Temp =             0x00000300;
      *ts0_Freq =             0x00000010;

      *ts1_Counter =          0x10000;
      *ts1_RD_Addr =    0x21FF0008;
```

```
	*ts1_WR_Addr =    0x21FF000C;
	*ts1_Temp =             0x00000300;
	*ts1_Freq =             0x00000010;

	*ts2_Counter =          0x10000;
	*ts2_RD_Addr =    0x21FF0010;
	*ts2_WR_Addr =    0x21FF0014;
	*ts2_Temp =             0x00000300;
	*ts2_Freq =             0x00000010;

	*ts3_Counter =          0x10000;
	*ts3_RD_Addr =    0x21FF0018;
	*ts3_WR_Addr =    0x21FF001C;
	*ts3_Temp =             0x00000300;
	*ts3_Freq =             0x00000010;
	// End initialize TS

	*ts_timer = 0xffffffff;
	*ts_timer = 0;

	//Bridged Memory matrix initialization
	for (i=0;i<MSIZE;i++)
		for (j=0;j<MSIZE;j++)
			if (i==j)
				COM[i*MSIZE+j+(PROC_NUM)] = i;
			else
				COM[i*MSIZE+j+(PROC_NUM)] = 0;

	/*COM[0] = NO_COMMAND;
	COM[1] = NO_COMMAND;
	COM[2] = NO_COMMAND;
	COM[3] = NO_COMMAND;*/

	while(1){

		/*COM[0] = MOVE_FROM_COM;
		while (COM[0] != 0){
		}



		COM[0] = MUL_MATRIX;
		COM[1] = MUL_MATRIX;
		COM[2] = MUL_MATRIX;
		COM[3] = MUL_MATRIX;

		while (COM[0] != 0 || COM[1] != 0 || COM[2] != 0 || COM[3]
!=0){}

		COM[0] = MOVE_TO_COM;

		while (COM[0] != 0){
		}*/

		while (*ts_timer == 0){

		}

		/*COM[0] = 0xFFFF0000;

		while (COM[0] != 0){
```

```
            }*/

            /*for (i=0;i<MSIZE;i++)
                  for (j=0;j<MSIZE;j++)
                        if (COM[i*MSIZE+j+(PROC_NUM*4)] != (j+1)*i)
                              while(1){}*/



            //calculate energies
            outputData[0] = *sniffer0_PM_WR;
            outputData[1] = *sniffer0_PM_RD;
            outputData[2] = *sniffer0_SM_WR;
            outputData[3] = *sniffer0_SM_RD;
            outputData[4] = *sniffer1_PM_WR;
            outputData[5] = *sniffer1_PM_RD;
            outputData[6] = *sniffer1_SM_WR;
            outputData[7] = *sniffer1_SM_RD;
            outputData[8] = *sniffer2_PM_WR;
            outputData[9] = *sniffer2_PM_RD;
            outputData[10] = *sniffer2_SM_WR;
            outputData[11] = *sniffer2_SM_RD;
            outputData[12] = *sniffer3_PM_WR;
            outputData[13] = *sniffer3_PM_RD;
            outputData[14] = *sniffer3_SM_WR;
            outputData[15] = *sniffer3_SM_RD;
            outputData[16] = *freq0;
            outputData[17] = *freq1;
            outputData[18] = *freq2;
            outputData[19] = *freq3;

            numTrans[0] =
*sniffer0_PM_WR+*sniffer0_PM_RD+*sniffer0_SM_WR+*sniffer0_SM_RD;
            numTrans[1] =
*sniffer1_PM_WR+*sniffer1_PM_RD+*sniffer1_SM_WR+*sniffer1_SM_RD;
            numTrans[2] =
*sniffer2_PM_WR+*sniffer2_PM_RD+*sniffer2_SM_WR+*sniffer2_SM_RD;
            numTrans[3] =
*sniffer3_PM_WR+*sniffer3_PM_RD+*sniffer3_SM_WR+*sniffer3_SM_RD;


            for (i=0;i<MSIZE;i++)
                  for (j=0;j<MSIZE;j++)
                        outputData[i*MSIZE+j+20] =
COM[i*MSIZE+j+(PROC_NUM)];

            for (i=0;i<MSIZE;i++)
            for (j=0;j<MSIZE;j++)
                  if (i==j)
                        COM[i*MSIZE+j+(PROC_NUM)] = i;
                  else
                        COM[i*MSIZE+j+(PROC_NUM)] = 0;

            //erase sniffers
            *sniffer0_PM_WR = 0xffffffff;
            *sniffer1_PM_WR = 0xffffffff;
            *sniffer2_PM_WR = 0xffffffff;
            *sniffer3_PM_WR = 0xffffffff;
            *sniffer0_PM_WR = 0x0;
            *sniffer1_PM_WR = 0x0;
```

```
		*sniffer2_PM_WR = 0x0;
		*sniffer3_PM_WR = 0x0;

		//send energies
		ptr_output = (char*)outputData;
		for (i=0;i<NUM_TRANS;i++){
			while(((*UART_SR)&0x00000008) != 0x0){}
			*UART_Transmit = *ptr_output;
			ptr_output++;
		}
		//waiting for the temperatures
		for (contTemp=0;contTemp<4;contTemp++){
			ptr_temp = &(temperatures[contTemp]);
			ptr_temp += 3;
			for (i=0;i<4;i++){
				while(((*UART_SR)&0x00000001) == 0x0){}

				*ptr_temp = *UART_Receive;
				ptr_temp--;
			}
		}
		//update temperatures
		*ts0_Temp = temperatures[0];
		*ts1_Temp = temperatures[1];
		*ts2_Temp = temperatures[2];
		*ts3_Temp = temperatures[3];

		/*TMU()*/
		switch (TMU_POL){
			case 0:
				if (temperatures[0] > THRESHOLD_H)
					*freq0 = CLK_BY_8;
				else
					if (temperatures[0] < THRESHOLD_L)
						*freq0 = CLK_BY_1;
				if (temperatures[1] > THRESHOLD_H)
					*freq1 = CLK_BY_8;
				else
					if (temperatures[1] < THRESHOLD_L)
						*freq1 = CLK_BY_1;
				if (temperatures[2] > THRESHOLD_H)
					*freq2 = CLK_BY_8;
				else
					if (temperatures[2] < THRESHOLD_L)
						*freq2 = CLK_BY_1;
				if (temperatures[3] > THRESHOLD_H)
					*freq3 = CLK_BY_8;
				else
					if (temperatures[3] < THRESHOLD_L)
						*freq3 = CLK_BY_1;

			break;
			case 1:
				if (temperatures[0] > THRESHOLD_H ||
temperatures[1] > THRESHOLD_H ||
				 temperatures[2] > THRESHOLD_H ||
temperatures[3] > THRESHOLD_H){
					*freq0 = CLK_BY_8;
					*freq1 = CLK_BY_8;
					*freq2 = CLK_BY_8;
					*freq3 = CLK_BY_8;
```

```
                        }
                        if (temperatures[0] < THRESHOLD_L &&
temperatures[1] < THRESHOLD_L &&
                              temperatures[2] < THRESHOLD_L &&
temperatures[3] < THRESHOLD_L){
                              *freq0 = CLK_BY_1;
                              *freq1 = CLK_BY_1;
                              *freq2 = CLK_BY_1;
                              *freq3 = CLK_BY_1;
                        }
                  break;
                  case 2:
                        //Processor 0 Thresholds
                        /*if (cooling[0] == 0){
                              if (temperatures[0] > THRESHOLD_H){
                                    *freq0 = CLK_BY_8;
                                    cooling[0] = 1;
                              }
                              else
                                    if (temperatures[0] > THRESHOLD_M)
                                          *freq0 = CLK_BY_4;
                                    else
                                          if (temperatures[0] >
THRESHOLD_L)
                                                *freq0 = CLK_BY_2;
                                          else
                                                *freq0 = CLK_BY_1;


                        //Processor 1 Thresholds
                              if (temperatures[1] > THRESHOLD_H){
                                    *freq1 = CLK_BY_8;
                                    cooling[0] = 1;
                              }
                              else
                                    if (temperatures[1] > THRESHOLD_M)
                                          *freq1 = CLK_BY_4;
                                    else
                                          if (temperatures[1] >
THRESHOLD_L)
                                                *freq1 = CLK_BY_2;
                                          else
                                                *freq1 = CLK_BY_1;

                        //Processor 2 Thresholds
                              if (temperatures[2] > THRESHOLD_H){
                                    *freq2 = CLK_BY_8;
                                    cooling[0] = 1;
                              }
                              else
                                    if (temperatures[2] > THRESHOLD_M)
                                          *freq2 = CLK_BY_4;
                                    else
                                          if (temperatures[2] >
THRESHOLD_L)
                                                *freq2 = CLK_BY_2;
                                          else
                                                *freq2 = CLK_BY_1;

                        //Processor 3 Thresholds
                              if (temperatures[3] > THRESHOLD_H){
```

```
                                    *freq3 = CLK_BY_8;
                                    cooling[0] = 1;
                              }
                              else
                                    if (temperatures[3] > THRESHOLD_M)
                                          *freq3 = CLK_BY_4;
                                    else
                                          if (temperatures[3] >
THRESHOLD_L)
                                                *freq3 = CLK_BY_2;
                                          else
                                                *freq3 = CLK_BY_1;

                  }else{
                        *freq0 = CLK_BY_8;
                        *freq1 = CLK_BY_8;
                        *freq2 = CLK_BY_8;
                        *freq3 = CLK_BY_8;
                        if (temperatures[0] < THRESHOLD_C &&
temperatures[1] < THRESHOLD_C &&
                              temperatures[2] < THRESHOLD_C &&
temperatures[3] < THRESHOLD_C){
                              cooling[0] = 0;
                              *freq0 = CLK_BY_1;
                              *freq1 = CLK_BY_1;
                              *freq2 = CLK_BY_1;
                              *freq3 = CLK_BY_1;
                        }
                  }*/
                  if (cooling == 0){
                        if (temperatures[0] > THRESHOLD_H ||
temperatures[1] > THRESHOLD_H ||
                              temperatures[2] > THRESHOLD_H ||
temperatures[3] > THRESHOLD_H){
                                    cooling = 1;
                                    *freq0 = CLK_BY_2;
                                    *freq1 = CLK_BY_2;
                                    *freq2 = CLK_BY_2;
                                    *freq3 = CLK_BY_2;
                        }else{
                                    *freq0 = CLK_BY_1;
                                    *freq1 = CLK_BY_1;
                                    *freq2 = CLK_BY_1;
                                    *freq3 = CLK_BY_1;
                        }
                  }else{
                        if (temperatures[0] < THRESHOLD_M &&
temperatures[1] < THRESHOLD_M &&
                              temperatures[2] < THRESHOLD_M &&
temperatures[3] < THRESHOLD_M){
                                    *freq0 = CLK_BY_4;
                                    *freq1 = CLK_BY_4;
                                    *freq2 = CLK_BY_4;
                                    *freq3 = CLK_BY_4;
                        }
                        if (temperatures[0] < THRESHOLD_L &&
temperatures[1] < THRESHOLD_L &&
                              temperatures[2] < THRESHOLD_L &&
temperatures[3] < THRESHOLD_L){
                                    *freq0 = CLK_BY_8;
                                    *freq1 = CLK_BY_8;
```

```
                                    *freq2 = CLK_BY_8;
                                    *freq3 = CLK_BY_8;
                        }
                        if (temperatures[0] < THRESHOLD_C &&
temperatures[1] < THRESHOLD_C &&
                              temperatures[2] < THRESHOLD_C &&
temperatures[3] < THRESHOLD_C){
                                    *freq0 = CLK_BY_1;
                                    *freq1 = CLK_BY_1;
                                    *freq2 = CLK_BY_1;
                                    *freq3 = CLK_BY_1;
                                    cooling = 0;
                        }
                  }



                  break;
                  case 3:
                        *freq0 = freqTable[0][contMod4];
                        *freq1 = freqTable[1][contMod4];
                        *freq2 = freqTable[2][contMod4];
                        *freq3 = freqTable[3][contMod4];
                  break;
                  case 4:
                        //Processor 0 Thresholds
                        /*if (temperatures[0] > THRESHOLD_H)
                              freqAnt[0] = CLK_BY_8;
                        else
                              if (temperatures[0] > THRESHOLD_M)
                                    freqAnt[0] = CLK_BY_4;
                              else
                                    if (temperatures[0] > THRESHOLD_L)
                                          freqAnt[0] = CLK_BY_2;
                                    else
                                          freqAnt[0] = CLK_BY_1;

                        //Processor 1 Thresholds
                        if (temperatures[1] > THRESHOLD_H)
                              freqAnt[1] = CLK_BY_8;
                        else
                              if (temperatures[1] > THRESHOLD_M)
                                    freqAnt[1] = CLK_BY_4;
                              else
                                    if (temperatures[1] > THRESHOLD_L)
                                          freqAnt[1] = CLK_BY_2;
                                    else
                                          freqAnt[1] = CLK_BY_1;

                        //Processor 2 Thresholds
                        if (temperatures[2] > THRESHOLD_H)
                              freqAnt[2] = CLK_BY_8;
                        else
                              if (temperatures[2] > THRESHOLD_M)
                                    freqAnt[2] = CLK_BY_4;
                              else
                                    if (temperatures[2] > THRESHOLD_L)
                                          freqAnt[2] = CLK_BY_2;
                                    else
```

```
                                    freqAnt[2] = CLK_BY_1;

                        //Processor 3 Thresholds
                        if (temperatures[3] > THRESHOLD_H)
                              freqAnt[3] = CLK_BY_8;
                        else
                              if (temperatures[3] > THRESHOLD_M)
                                    freqAnt[3] = CLK_BY_4;
                              else
                                    if (temperatures[3] > THRESHOLD_L)
                                          freqAnt[3] = CLK_BY_2;
                                    else
                                                freqAnt[3] =
CLK_BY_1;*/

                  if (cooling == 0){
                        if (temperatures[0] > THRESHOLD_H ||
temperatures[1] > THRESHOLD_H ||
                              temperatures[2] > THRESHOLD_H ||
temperatures[3] > THRESHOLD_H){
                                    cooling = 1;
                                    freqAnt[0] = CLK_BY_2;
                                    freqAnt[1] = CLK_BY_2;
                                    freqAnt[2] = CLK_BY_2;
                                    freqAnt[3] = CLK_BY_2;
                        }else{
                                    freqAnt[0] = CLK_BY_1;
                                    freqAnt[1] = CLK_BY_1;
                                    freqAnt[2] = CLK_BY_1;
                                    freqAnt[3] = CLK_BY_1;
                        }
                  }else{
                        if (temperatures[0] < THRESHOLD_M &&
temperatures[1] < THRESHOLD_M &&
                              temperatures[2] < THRESHOLD_M &&
temperatures[3] < THRESHOLD_M){
                                    freqAnt[0] = CLK_BY_4;
                                    freqAnt[1] = CLK_BY_4;
                                    freqAnt[2] = CLK_BY_4;
                                    freqAnt[3] = CLK_BY_4;
                        }
                        if (temperatures[0] < THRESHOLD_L &&
temperatures[1] < THRESHOLD_L &&
                              temperatures[2] < THRESHOLD_L &&
temperatures[3] < THRESHOLD_L){
                                    freqAnt[0] = CLK_BY_8;
                                    freqAnt[1] = CLK_BY_8;
                                    freqAnt[2] = CLK_BY_8;
                                    freqAnt[3] = CLK_BY_8;
                        }
                        if (temperatures[0] < THRESHOLD_C &&
temperatures[1] < THRESHOLD_C &&
                              temperatures[2] < THRESHOLD_C &&
temperatures[3] < THRESHOLD_C){
                                    freqAnt[0] = CLK_BY_1;
                                    freqAnt[1] = CLK_BY_1;
                                    freqAnt[2] = CLK_BY_1;
                                    freqAnt[3] = CLK_BY_1;
                                    cooling = 0;
                        }
                  }
```

```
		if (numTrans[0] == 0)
			*freq0 = CLK_BY_8;
		else
			*freq0 = freqAnt[0];
		if (numTrans[1] == 0)
			*freq1 = CLK_BY_8;
		else
			*freq1 = freqAnt[1];
		if (numTrans[2] == 0)
			*freq2 = CLK_BY_8;
		else
			*freq2 = freqAnt[2];
		if (numTrans[3] == 0)
			*freq3 = CLK_BY_8;
		else
			*freq3 = freqAnt[3];
	break;
	case 5:
		if (temperatures[0] > THRESHOLD_H)
			freqAnt[0] = CLK_BY_8;
		else
			if (temperatures[0] < THRESHOLD_L)
				freqAnt[0] = CLK_BY_1;
		if (temperatures[1] > THRESHOLD_H)
			freqAnt[1] = CLK_BY_8;
		else
			if (temperatures[1] < THRESHOLD_L)
				freqAnt[1] = CLK_BY_1;
		if (temperatures[2] > THRESHOLD_H)
			freqAnt[2] = CLK_BY_8;
		else
			if (temperatures[2] < THRESHOLD_L)
				freqAnt[2] = CLK_BY_1;
		if (temperatures[3] > THRESHOLD_H)
			freqAnt[3] = CLK_BY_8;
		else
			if (temperatures[3] < THRESHOLD_L)
				freqAnt[3] = CLK_BY_1;

		if (numTrans[0] == 0)
			*freq0 = CLK_BY_8;
		else
			*freq0 = freqAnt[0];
		if (numTrans[1] == 0)
			*freq1 = CLK_BY_8;
		else
			*freq1 = freqAnt[1];
		if (numTrans[2] == 0)
			*freq2 = CLK_BY_8;
		else
			*freq2 = freqAnt[2];
		if (numTrans[3] == 0)
			*freq3 = CLK_BY_8;
		else
			*freq3 = freqAnt[3];
	break;
	case 6:
		//Processor 0 Thresholds
		if (temperatures[0] > THRESHOLD_H)
			freqAnt[0] = CLK_BY_8;
```

```
else
      if (temperatures[0] > THRESHOLD_M)
            freqAnt[0] = CLK_BY_4;
      else
            if (temperatures[0] > THRESHOLD_L)
                  freqAnt[0] = CLK_BY_2;
            else
                  freqAnt[0] = CLK_BY_1;

//Processor 1 Thresholds
if (temperatures[1] > THRESHOLD_H)
      freqAnt[1] = CLK_BY_8;
else
      if (temperatures[1] > THRESHOLD_M)
            freqAnt[1] = CLK_BY_4;
      else
            if (temperatures[1] > THRESHOLD_L)
                  freqAnt[1] = CLK_BY_2;
            else
                  freqAnt[1] = CLK_BY_1;

//Processor 2 Thresholds
if (temperatures[2] > THRESHOLD_H)
      freqAnt[2] = CLK_BY_8;
else
      if (temperatures[2] > THRESHOLD_M)
            freqAnt[2] = CLK_BY_4;
      else
            if (temperatures[2] > THRESHOLD_L)
                  freqAnt[2] = CLK_BY_2;
            else
                  freqAnt[2] = CLK_BY_1;

//Processor 3 Thresholds
if (temperatures[3] > THRESHOLD_H)
      freqAnt[3] = CLK_BY_8;
else
      if (temperatures[3] > THRESHOLD_M)
            freqAnt[3] = CLK_BY_4;
      else
            if (temperatures[3] > THRESHOLD_L)
                  freqAnt[3] = CLK_BY_2;
            else
                        freqAnt[3] = CLK_BY_1;
if (numTrans[0] == 0)
      *freq0 = CLK_BY_8;
else
      *freq0 = freqAnt[0];
if (numTrans[1] == 0)
      *freq1 = CLK_BY_8;
else
      if (freqAnt[1] > freqAnt[0]){
            cont[0]++;
            *freq1 = freqAnt[0];
      }else
            *freq1 = freqAnt[1];
if (numTrans[2] == 0)
      *freq2 = CLK_BY_8;
else
      if (freqAnt[2] > freqAnt[0]){
            cont[1]++;
```

```
                        *freq2 = freqAnt[0];
                  }else
                        *freq2 = freqAnt[2];
            if (numTrans[3] == 0)
                  *freq3 = CLK_BY_8;
            else
                  if (freqAnt[3] > freqAnt[0]){
                        cont[2]++;
                        *freq3 = freqAnt[0];
                  }else
                        *freq3 = freqAnt[3];

      break;
      case 7:
                  //Processor 0 Thresholds
            /*if (temperatures[0] > THRESHOLD_H)
                  freqAnt[0] = CLK_BY_8;
            else
                  if (temperatures[0] > THRESHOLD_M)
                        freqAnt[0] = CLK_BY_4;
                  else
                        if (temperatures[0] > THRESHOLD_L)
                              freqAnt[0] = CLK_BY_2;
                        else
                              freqAnt[0] = CLK_BY_1;
*/
            if (cooling == 0){
            if (temperatures[0] > THRESHOLD_H){
                        cooling = 1;
                        freqAnt[0] = CLK_BY_2;
            }else{
                        freqAnt[0] = CLK_BY_1;
            }
      }else{
            if (temperatures[0] < THRESHOLD_M){
                        freqAnt[0] = CLK_BY_4;
            }
            if (temperatures[0] < THRESHOLD_L){
                        freqAnt[0] = CLK_BY_8;
            }
            if (temperatures[0] < THRESHOLD_C){
                        freqAnt[0] = CLK_BY_1;
                        cooling = 0;
            }
      }

            if (freqAnt[0] == CLK_BY_1)
                  freqMitad = CLK_BY_2;
            else if (freqAnt[0] == CLK_BY_2)
                  freqMitad = CLK_BY_4;
            else
                  freqMitad = CLK_BY_8;

            //Processor 1
            /*if (contMod4 != 3)
                  freqAnt[1] = freqAnt[0];
            else
                  freqAnt[1] = freqMitad;*/
            //Processor 1
            if (contMod4 < 2)
                  freqAnt[1] = freqAnt[0];
```

```
                        else
                              freqAnt[1] = freqMitad;
                        //Processor 2
                        if (contMod4 == 0)
                              freqAnt[2] = freqAnt[0];
                        else
                              freqAnt[2] = freqMitad;
                        //Processor 3
                        freqAnt[3] = freqMitad;

                        if (numTrans[0] == 0)
                              *freq0 = CLK_BY_8;
                        else
                              *freq0 = freqAnt[0];
                        if (numTrans[1] == 0)
                              *freq1 = CLK_BY_8;
                        else
                              *freq1 = freqAnt[1];
                        if (numTrans[2] == 0)
                              *freq2 = CLK_BY_8;
                        else
                              *freq2 = freqAnt[2];
                        if (numTrans[3] == 0)
                              *freq3 = CLK_BY_8;
                        else
                              *freq3 = freqAnt[3];
                  break;
                  default:
                        *freq0 = CLK_BY_1;
                        *freq1 = CLK_BY_1;
                        *freq2 = CLK_BY_1;
                        *freq3 = CLK_BY_1;
                  }


            contMod4++;
            contMod4 %= 4;
            //put on the emulated system
            *ts_timer = 0xffffffff;
            *ts_timer = 0;



            //printf("\033[H\033[J"); //Clear Screen

      }

   /*
    * MemoryTest routine will not be run for the memory at
    * 0x00000000 (dlmb_cntlr)
    * because it is being used to hold a part of this application
program
    */

   /*
    * Disable cache and reinitialize it so that other
    * applications can be run with no problems
    */
   #if XPAR_MICROBLAZE_0_USE_DCACHE
      microblaze_disable_dcache();
```

```
      microblaze_init_dcache_range(0,
XPAR_MICROBLAZE_0_DCACHE_BYTE_SIZE);
   #endif

   #if XPAR_MICROBLAZE_0_USE_ICACHE
      microblaze_disable_icache();
      microblaze_init_icache_range(0,
XPAR_MICROBLAZE_0_CACHE_BYTE_SIZE);
   #endif


   //print("-- Exiting main() --\r\n");
   return 0;
}
```

# Código Processor Element 1

```
#include "xparameters.h"
#include "xio.h"

//master number
#define MNUMBER 0

#define PROC_NUM 4

//define matrix size and lines to process
#define MSIZE 32
#define STARTL 0
#define ENDL 8

//define adresses
#define PM  0x00000000
#define SHM 0x19000000
#define COM 0x21000000

#define BRIDGEREG    XPAR_OPB2OCP_TSBRIDGE_M0_BASEADDR
#define BRIDGEOFFSET XPAR_OPB2OCP_TSBRIDGE_M0_AR0_BASEADDR

void mul_matrix(void);
void mul_matrix_num(int num);
void move_rez(void);
void move_rez_par(int start, int end);
int wait_command(void);
void move_matrix(void);
void init_coeff_matrix(void);
void clear_command(void);
void put_command(void);
void wait_all_procesor(void);
void wait_processor(int num);

int main(void){
      int opc,veces;
      unsigned int num = 0;


      init_coeff_matrix();

      while(1){
            move_matrix();
            put_command();
            mul_matrix_num(100);
            /*for (veces=0;veces<1000;veces++)
                  move_rez();*/
            for (veces=0;veces<2000;veces++)
                  move_rez_par(0,8);
            wait_processor(1);
            for (veces=0;veces<2000;veces++)
                  move_rez_par(8,16);
            wait_processor(2);
            for (veces=0;veces<2000;veces++)
                  move_rez_par(16,24);
            wait_processor(3);
            for (veces=0;veces<2000;veces++)
```

```
                move_rez_par(24,32);
        }

        return 1;
}

void mul_matrix(void){
        volatile long *d = (long *) (BRIDGEOFFSET + (MSIZE * MSIZE *
4));
        volatile long *s = (long *) (BRIDGEOFFSET);
        int i, j, k;
        long tmp, coeff, val;

        for(i = STARTL; i < ENDL; i++){
                for(k = 0; k < MSIZE; k++){
                        tmp = 0;
                        for(j = 0; j < MSIZE; j++){
                                XIo_Out32(BRIDGEREG, PM);
                                coeff = s[(i * MSIZE) + j];
                                XIo_Out32(BRIDGEREG, SHM);
                                val = s[(j * MSIZE) + k];
                                tmp += coeff * val;
                        }
                        XIo_Out32(BRIDGEREG, SHM);
                        d[(i * MSIZE) + k] = tmp;
                }
        }
        /*
        XIo_Out32(BRIDGEREG, SHM);
        for(i = 0; i < MSIZE; i++)
                for(j = 0; j < MSIZE; j++)
                        d[(i * MSIZE) + j] = s[(i * MSIZE) + j];
        */
}
void mul_matrix_num(int num){
        volatile long *d = (long *) (BRIDGEOFFSET + (MSIZE * MSIZE *
4));
        volatile long *s = (long *) (BRIDGEOFFSET);
        int i, j, k,aux;
        long tmp, coeff, val;


        for (aux = 0; aux < num; aux++)
                for(i = STARTL; i < ENDL; i++){
                        for(k = 0; k < MSIZE; k++){
                                tmp = 0;
                                for(j = 0; j < MSIZE; j++){
                                        XIo_Out32(BRIDGEREG, PM);
                                        coeff = s[(i * MSIZE) + j];
                                        XIo_Out32(BRIDGEREG, SHM);
                                        val = s[(j * MSIZE) + k];
                                        tmp += coeff * val;
                                }
                                XIo_Out32(BRIDGEREG, SHM);
                                d[(i * MSIZE) + k] = tmp;
                        }
                }
}
void move_rez(void){
        volatile long *d = (long *) (BRIDGEOFFSET + (PROC_NUM * 4));
```

```
	volatile long *s = (long *) (BRIDGEOFFSET + (MSIZE * MSIZE *
4));
	int i, j;
	long tmp;

	for(i = 0; i < MSIZE; i++)
		for(j = 0; j < MSIZE; j++){
			XIo_Out32(BRIDGEREG, SHM);
			tmp = *s;
			XIo_Out32(BRIDGEREG, COM);
			*d = tmp;
			s++;
			d++;
		}
}

void move_rez_par(int start, int end){
volatile long *d = (long *) (BRIDGEOFFSET + (PROC_NUM * 4));
	volatile long *s = (long *) (BRIDGEOFFSET + (MSIZE * MSIZE *
4));
	int i, j;
	long tmp;

	for(i = start; i < end; i++)
		for(j = 0; j < MSIZE; j++){
			XIo_Out32(BRIDGEREG, SHM);
			tmp = *s;
			XIo_Out32(BRIDGEREG, COM);
			*d = tmp;
			s++;
			d++;
		}
}

void move_matrix(void){
	volatile long *s = (long *) (BRIDGEOFFSET + (PROC_NUM * 4));
	volatile long *d = (long *) BRIDGEOFFSET;
	int i, j;
	long tmp;

	for(i = 0; i < MSIZE; i++)
		for(j = 0; j < MSIZE; j++){
			XIo_Out32(BRIDGEREG, COM);
			tmp = *s;
			XIo_Out32(BRIDGEREG, SHM);
			*d = tmp;
			s++;
			d++;
		}

}

void clear_command(void){
	volatile long *p = (long *) (BRIDGEOFFSET + (MNUMBER * 4));

	XIo_Out32(BRIDGEREG, COM);
	*p = 0;
}

int wait_command(void){
	volatile long *p = (long *) (BRIDGEOFFSET + (MNUMBER * 4));
```

```
	long i;
	int j;

	XIo_Out32(BRIDGEREG, COM);

	while((i = *p) == 0x00000000)
		for(j = 0; j < 100; j++);
	if(i == 0x0000FFFF)
		return 0;
	if(i == 0xFFFF0000)
		return 1;
	return 2;

}

void put_command(void){
	volatile long *p = (long *) (BRIDGEOFFSET);
	int i;

	XIo_Out32(BRIDGEREG, COM);
	for (i=0;i<4;i++){
		*p=0x0000FFFF;
		p++;
	}
}

void wait_all_procesor(void){
	volatile long *p = (long *) (BRIDGEOFFSET);
	int j;

	XIo_Out32(BRIDGEREG, COM);
	while(p[1] != 0x0 || p[2] != 0x0 || p[3] != 0x0)
		for(j = 0; j < 100; j++);

}

void wait_processor(int num){
volatile long *p = (long *) (BRIDGEOFFSET);
	int j;

	XIo_Out32(BRIDGEREG, COM);
	while(p[num] != 0x0)
		for(j = 0; j < 100; j++);

}

void init_coeff_matrix(void){
	volatile long *p = (long *) BRIDGEOFFSET;
	int i, j;

	XIo_Out32(BRIDGEREG, PM);

	for(i = 0; i < MSIZE; i++)
		for(j = 0; j < MSIZE; j++)
			p[(i * MSIZE) + j] = (i * MSIZE) + j;
}
```

## Código Processor Element 2,3 y 4

```
#include "xparameters.h"
#include "xio.h"

//master number
#define MNUMBER 1

#define PROC_NUM 4

//define matrix size and lines to process
#define MSIZE 32
#define STARTL 8
#define ENDL 16

//define adresses
#define PM  0x01000000
#define SHM 0x19000000
#define COM 0x21000000

#define BRIDGEREG    XPAR_OPB2OCP_TSBRIDGE_M1_BASEADDR
#define BRIDGEOFFSET XPAR_OPB2OCP_TSBRIDGE_M1_AR0_BASEADDR

void mul_matrix(void);
void mul_matrix_num(int num);
void move_rez(void);
int wait_command(void);
void move_matrix(void);
void init_coeff_matrix(void);
void clear_command(void);

int main(void){
      int opc;


      init_coeff_matrix();

      while(1){
            opc = wait_command();
            switch (opc){
                  case 0:
                        mul_matrix_num(100);
                  break;
                  case 1:
                        move_rez();
                  break;
                  case 2:
                        move_matrix();
                  break;
            }
            clear_command();
      }

      return 1;
}

void mul_matrix(void){
      volatile long *d = (long *) (BRIDGEOFFSET + (MSIZE * MSIZE *
4));
      volatile long *s = (long *) (BRIDGEOFFSET);
      int i, j, k;
```

```
	long tmp, coeff, val;

	for(i = STARTL; i < ENDL; i++){
		for(k = 0; k < MSIZE; k++){
			tmp = 0;
			for(j = 0; j < MSIZE; j++){
				XIo_Out32(BRIDGEREG, PM);
				coeff = s[(i * MSIZE) + j];
				XIo_Out32(BRIDGEREG, SHM);
				val = s[(j * MSIZE) + k];
				tmp += coeff * val;
			}
			XIo_Out32(BRIDGEREG, SHM);
			d[(i * MSIZE) + k] = tmp;
		}
	}
	/*
	XIo_Out32(BRIDGEREG, SHM);
	for(i = 0; i < MSIZE; i++)
		for(j = 0; j < MSIZE; j++)
			d[(i * MSIZE) + j] = s[(i * MSIZE) + j];
	*/
}

void mul_matrix_num(int num){
	volatile long *d = (long *) (BRIDGEOFFSET + (MSIZE * MSIZE *
4));
	volatile long *s = (long *) (BRIDGEOFFSET);
	int i, j, k,aux;
	long tmp, coeff, val;


	for (aux = 0; aux < num; aux++)
		for(i = STARTL; i < ENDL; i++){
			for(k = 0; k < MSIZE; k++){
				tmp = 0;
				for(j = 0; j < MSIZE; j++){
					XIo_Out32(BRIDGEREG, PM);
					coeff = s[(i * MSIZE) + j];
					XIo_Out32(BRIDGEREG, SHM);
					val = s[(j * MSIZE) + k];
					tmp += coeff * val;
				}
				XIo_Out32(BRIDGEREG, SHM);
				d[(i * MSIZE) + k] = tmp;
			}
		}
}

void move_rez(void){
	volatile long *d = (long *) (BRIDGEOFFSET + (PROC_NUM * 4));
	volatile long *s = (long *) (BRIDGEOFFSET + (MSIZE * MSIZE *
4));
	int i, j;
	long tmp;

	for(i = 0; i < MSIZE; i++)
		for(j = 0; j < MSIZE; j++){
			XIo_Out32(BRIDGEREG, SHM);
			tmp = *s;
			XIo_Out32(BRIDGEREG, COM);
```

```
                *d = tmp;
                s++;
                d++;
            }
}

void move_matrix(void){
      volatile long *s = (long *) (BRIDGEOFFSET + (PROC_NUM * 4));
      volatile long *d = (long *) BRIDGEOFFSET;
      int i, j;
      long tmp;

      for(i = 0; i < MSIZE; i++)
            for(j = 0; j < MSIZE; j++){
                XIo_Out32(BRIDGEREG, COM);
                tmp = *s;
                XIo_Out32(BRIDGEREG, SHM);
                *d = tmp;
                s++;
                d++;
            }

}

void clear_command(void){
      volatile long *p = (long *) (BRIDGEOFFSET + (MNUMBER * 4));

      XIo_Out32(BRIDGEREG, COM);
      *p = 0;
}

int wait_command(void){
      volatile long *p = (long *) (BRIDGEOFFSET + (MNUMBER * 4));
      long i;
      int j;

      XIo_Out32(BRIDGEREG, COM);

      while((i = *p) == 0x00000000)
            for(j = 0; j < 100; j++);
      if(i == 0x0000FFFF)
            return 0;
      if(i == 0xFFFF0000)
            return 1;
      return 2;

}

void init_coeff_matrix(void){
      volatile long *p = (long *) BRIDGEOFFSET;
      int i, j;

      XIo_Out32(BRIDGEREG, PM);

      for(i = 0; i < MSIZE; i++)
            for(j = 0; j < MSIZE; j++)
                p[(i * MSIZE) + j] = (i * MSIZE) + j;
}
```